



City Research Online

City, University of London Institutional Repository

Citation: Procopiou, A. (2021). ForChaosLR: A Lightweight Multi-Layer IDS to detect application layer DDoS attacks in the smart home-smart grid network. (Unpublished Doctoral thesis, City, University of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/27616/>

Link to published version:

Copyright: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

Reuse: Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

ForChaosLR: A Lightweight Multi-Layer IDS to detect application layer DDoS attacks in the smart home-smart grid network



Andria Procopiou

Department of Computer Science, School of Computer Science,
Mathematics and Engineering
City, University of London

This dissertation is submitted for the degree of
Doctor of Philosophy

December 2021

"...but in all chaos there's calculation ..." - (Glory and Gore by Lorde, 2014)

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Andria Procopiou

December 2021

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Thomas Chen for his unwavering help and support. His sharing of knowledge and experience, words of encouragement and full support have been catalytic in me being shaped to the researcher I am today. His trust in me has enabled me to fully explore and discover my personal academic interests and passions and ultimately maturing as a researcher. Furthermore, I would like to also thank my second supervisor Dr. Eduardo Alonso, for his words of encouragement and his valuable support.

I am deeply grateful to my parents, Sylvia and Panayiotis, for their endless and unconditional love and support. Their teachings on kindness, transparency, determination, honesty, wisdom, courage and bravery have been essential on this journey. Thank you for everything. I would also like to thank my sister Nikoletta, for her love, encouragement, support and willingness to have long late-night conversations with me. To my aunt and uncle, Mary and Charalampos, for sharing my passion for engineering and computer science. To my late grandparents, Nikki and Jason, who always believed I could achieve great things. To my cousin, Vicky, who made sure to share a good laugh with me and also made sure to remind me that success is in our blood. To my family friends for becoming my second family.

To all of my friends, you know who you are, for their love, support, encouragement, kind words, surprise visits and for always reminding of me what I am capable of. To all of my friends from the photography club for their valuable support.

To D.H. for her valuable help when I needed it the most. To M.B. for being the best role model anyone could ever ask for. To C.K. for making me realise that anyone can make a great difference and help others.

To Harry, Ron and Hermione for keeping me company through their stories all these years. To all my teachers throughout the years whose teaching, support and values have been essential for my own journey in education and life.

To London, for being the best city anyone could live.

Abstract

Worldwide issues such as overpopulation, environmental pollution, emissions exhaustion, global warming, energy waste, health problems, public transport congestion and overcrowding, the concept of a smart city has emerged. A smart city's main objective is to improve the life quality of its citizens, promote sustainability and protect the environment. For the realisation of this vision, a set of different technologies are adopted that can transform the city's components into "smart entities". The most catalytic technology for this is the integration of the Internet of Things (IoT). Through IoT mechanical/electrical entities are able to communicate with each other directly and exchange useful data to their functioning, are connected to the Internet and can be controlled remotely and collect data from the environment they are deployed at.

Their multiple benefits are highlighted through a plethora of different networks and systems, with a notable example being the smart home-smart grid network. This interaction between the two networks is one of the most beneficial to the smart city as it helps residential owners in their tasks and improves their quality of life. Furthermore, the two-way communication between the smart home and the smart grid offers the smart city citizens the opportunity to manage their electricity consumption efficiently and in an environmentally friendly way, decrease their electricity bill, be part of energy saving up reward programmes, have their smart appliances and intelligent devices being monitored and maintained remotely by relevant third-party services and even exchange energy to the smart grid, provided they have energy distributed services installed in their premises.

This tight and high inter-connectivity between the two networks can be maliciously used to perform cross-network attacks, especially from being initiated from the smart home side where the security measures might be weak or non-existent. In the past, there have been numerous security incidents where IoT devices have been easily compromised due to their lack of proper security solutions deployed. Notable examples of such incidents are the Mirai botnet, Aida, Bashlite and IIRCTelnet.

One of the most popular attacks which is likely to cause the most immediate effect is a Distributed Denial of Service (DDoS) attack. Of particular interest are application layer DDoS attacks, specifically flooding and slow-rate, as they can cause more damage to the

target. They require less resources and are much stealthier and similar to legitimate traffic. Therefore, such attacks are harder to be detected by an intrusion detection system (IDS).

For the detection of such attacks, multiple IDSs have been proposed, most of them consisting machine learning algorithms. Although the proposed systems achieved remarkable results they are high in computational complexity and resources, such as training time, memory usage, CPU power and training data. As a result, they can cause computational burden and exhaust the resources of IoT devices. Therefore, a proposed IDS solution must be as lightweight as possible so it does not cause any computational burden. The majority of lightweight approaches consist of constructing a baseline approach, based on legitimate behaviour observed, and flagging anything that falls out of this baseline. Such an approach could potentially raise a high number of false alarms. Therefore, there should be an additional layer that cooperates with the lightweight detection layer in regulating the false alarms but not to an extent it increases the complexity significantly. It is important to define an effective trade-off between overall accuracy and resource constraints as many of the IoT devices deployed in the smart home-smart grid network so the IDS is accurate in classifying incoming traffic correctly but also not exhausting the resources of the device.

The thesis proposes a novel multi-layer IDS system for application layer flooding and slow-rate DDoS attacks in the smart home-smart grid network that aims to accurately detect such attacks, achieve a low false positive rate and maintain its low complexity in computational resources. On the first layer we implemented a combination of simple exponential smoothing forecasting algorithm with lyapunov exponents measuring chaotic behaviour mechanism. This combination is lightweight in resources, training time and size and has been proved to be effective against such attacks. On the second layer we deployed logistic regression which is more sophisticated, especially in regulating false positives, but still lighter than heavier machine learning algorithms such as neural networks, random forest, Bayesian networks and support vector machines.

We implemented a simulation of a smart home-smart grid network in NS-3 network simulator and generated enough data to construct a dataset. Our proposed system, ForChaosLR multi-layer IDS, was evaluated using the dataset constructed and compared to other notable and popular machine learning algorithms. Furthermore, we evaluated ForChaosLR compared to the chosen set of machine learning algorithms using other real-world datasets to evaluate its effectiveness in further.

Proceeding, we evaluated its accuracy in classifying incoming malicious traffic to a specific attack category and monitored its computational resources (memory usage, CPU power and training/testing time) using a Raspberry Pi 3 model B+. Once again, ForChaosLR's

results were compared to the equivalent results achieved from the machine learning algorithms chosen.

Through the different sets of experiments conducted, we demonstrated that ForChaosLR multi-layer IDS is a suitable lightweight IDS system for a resource-constrained network, such as the smart home-smart grid, that is accurate in detecting the application layer flooding and slow-rate DDoS attacks without overstressing the Raspberry Pi 3 model B+ and its resources, specifically memory, CPU power. ForChaosLR is also successful in classifying application layer DDoS attacks to their respective categories and attack types.

Table of contents

List of figures	xix
List of tables	xxiii
1 Introduction	1
1.1 Problem Statement and Motivation	1
1.1.1 Smart City and IoT	1
1.1.2 Smart Home-Smart Grid network and its benefits	4
1.1.3 Smart Home-Smart Grid network Security Concerns	6
1.2 Objectives	10
1.3 Contributions	11
1.4 Published Work	13
1.5 Thesis Outline	14
2 Background Knowledge	17
2.1 Overview	17
2.2 Smart Home-Smart Grid Architecture	17
2.2.1 Smart Grid Architecture	17
2.2.2 Smart Home Architecture	20
2.2.3 Smart Home-Smart Grid Architecture	20
2.2.4 Smart Home-Smart Grid Network Communications Protocols	22
2.2.5 Smart Home-Smart Grid Network Devices Specifications	25
2.3 Smart Home-Smart Grid Network Security	26
2.3.1 IoT devices Security Issues and Vulnerabilities in the Smart Home-Smart Grid Network	26
2.3.2 IoT Attacks Taxonomy	29
2.4 Intrusion Detection Systems	37
2.4.1 IDS Taxonomy	38

2.5	IDS Detection Techniques	39
2.5.1	Time-Series Forecasting Approaches	39
2.5.2	Chaos Theory	41
2.5.3	Introduction	41
2.5.4	Properties of chaotic systems	41
2.5.5	Lyapunov exponent in measuring sensitivity to initial conditions . .	42
2.5.6	Lyapunov Exponents in DDoS detection	43
2.5.7	Machine Learning Techniques	44
2.5.8	Regression Analysis used in Machine Learning	46
2.6	Dataset Imbalance Prevention Techniques	49
2.7	Summary	50
3	Literature Review	51
3.1	Overview	51
3.2	IDS systems studies on application layer DDos attacks	53
3.3	IDS Studies using machine learning and other A.I. techniques	55
3.3.1	IDS Studies using machine learning and other A.I. techniques for the smart home and/or the smart grid	55
3.3.2	IDS Studies using machine learning and other A.I. techniques for other resource-constrained networks	57
3.3.3	IDS Studies using A.I. lightweight techniques	61
3.4	Multi-layer/multi-detection IDS studies	63
3.5	Overall Critique and Current Literature Limitations and Gaps	70
4	ForChaosLR Algorithmic Design	73
4.1	Overview	73
4.2	Feature Selection	74
4.3	Motivation for a multi-layer approach IDS	78
4.4	ForChaos First-Layer Modelling	82
4.4.1	Forecasting to Predict Future Network Behaviour of the System . .	82
4.4.2	Chaos Theory for Error Analysis	84
4.4.3	ForChaos First-Layer Training	86
4.5	LR second-layer Modelling	88
4.5.1	LR second-layer Training for Binary Classification	88
4.5.2	LR second-layer training for Multi-class Classification	89
4.6	ForChaos First-Layer and LR Second-Layer Detection Model	90
4.7	ForChaos First-Layer and LR Second-Layer Multi-Attack Detection Model	92

4.8	Summary	93
5	Methodology of Experiments	95
5.1	Overview	95
5.2	Datasets used for ForChaosLR multi-layer IDS Evaluation	97
5.2.1	NS-3 Binary Datasets	97
5.2.2	Real World Binary Datasets	99
5.2.3	NS-3 and CICIDS 2017 Multi-Class Datasets	102
5.3	Network Simulator 3 (NS-3) Simulator	104
5.4	Smart Home-Smart Grid Network Simulation	106
5.4.1	Smart Home-Smart Grid Network Critical Nodes	106
5.4.2	Smart Home-Smart Grid Network Protocols	107
5.4.3	Smart Home-Smart Grid Network Simulation in NS-3	110
5.4.4	Smart Home-Smart Grid network legitimate traffic generation	113
5.4.5	DDoS Simulation	114
5.5	Weka Toolbox and Weka Libraries	118
5.6	Summary	120
6	ForChaosLR Evaluation using the NS-3 Binary Dataset	121
6.1	Overview	121
6.2	Evaluation Metrics	122
6.3	Scenarios Independent Variables	123
6.3.1	Window Size w	124
6.3.2	Constant α	125
6.3.3	Behaviour Type	126
6.3.4	Attack Intensity	127
6.3.5	Attack Duration	127
6.3.6	Number of Attackers	127
6.3.7	Number of New Nodes	128
6.3.8	Number of Applications	128
6.3.9	Second Attack Performed	128
6.4	ForChaosLR multi-layer Parameters to be Optimised in Training	129
6.5	Training of ForChaos first-layer	129
6.5.1	Overview	129
6.5.2	Evaluation of a and window size w	131
6.5.3	Evaluation of ForChaos first-layer Training Time	134
6.6	ForChaos first-layer Testing and Evaluation	137

6.6.1	Overview	137
6.6.2	ForChaos against Constant Rate DDoS Attacks Experiments	137
6.6.3	Results Analysis and Discussion	137
6.6.4	ForChaos against Increasing Rate DDoS Attacks Experiments . . .	140
6.6.5	Results Analysis and Discussion	141
6.6.6	ForChaos against Mixed (Flooding and Slow-Rate) Attack Experiments	143
6.6.7	ForChaos against Flash Crowd Experiments	145
6.6.8	ForChaos against Restart Attack Experiments	151
6.6.9	Summary of ForChaos first-layer Evaluation	156
6.7	Training Procedure of LR Second-Layer	157
6.7.1	Motivation	157
6.7.2	Experimental Set Up	159
6.7.3	Results Analysis and Discussion	160
6.8	LR second-layer Evaluation	162
6.8.1	Overview	162
6.9	Summary of ForChaosLR against NS-3 Dataset	163
6.10	Evaluation of ForChaosLR against popular machine learning algorithms . .	165
6.10.1	Overview	165
6.10.2	Experimental Set Up	166
6.10.3	Results Analysis and Discussion	166
6.11	Summary	174
7	ForChaosLR Accuracy Evaluation using Real-World Datasets	175
7.1	Overview	175
7.2	Evaluation of ForChaosLR against ISCXIDS 2012 and CICIDS 2017 Datasets	177
7.2.1	Motivation	177
7.2.2	Experimental Setup Details	178
7.2.3	Results Analysis and Discussion of ISCXIDS 2012	179
7.2.4	ForChaosLR ISCXIDS 2012 Flooding Day Results	179
7.2.5	ForChaosLR ISCXIDS 2012 Normal Day Results	180
7.2.6	ForChaosLR compared to ML Algorithms against ISCXIDS 2012 Dataset	188
7.2.7	Evaluation of ForChaosLR against CICIDS 2017 Dataset	196
7.2.8	ForChaosLR compared to ML Algorithms against CICIDS 2017 Dataset	199
7.3	Evaluation of ForChaosLR using Smart Home Dataset	207
7.3.1	Motivation	207

7.3.2	Experimental Setup Details	207
7.3.3	Results Analysis and Discussion	208
7.4	Summary	209
8	ForChaosLR IDS Accuracy Evaluation using Multi-Class Datasets	215
8.1	Overview and Motivation	215
8.2	Experimental Setup Details for multi-attack datasets	216
8.3	Results Analysis for NS-3 multi-class dataset	217
8.4	Results Analysis for CICSIDS2017 Single-Category Multi-class Dataset . .	223
8.5	Results Analysis for CICSIDS2017 Multi-Category Multi-Class Dataset . .	229
8.6	Summary	233
9	ForChaosLR evaluation in Raspberry Pi 3 model B+	251
9.1	Overview	251
9.2	Results Analysis and Discussion for Binary Datasets	252
9.2.1	Results Analysis and Discussion for NS-3 Datasets	252
9.2.2	Results Analysis and Discussion for ISCXIDS 2012 Dataset	256
9.2.3	Results Analysis and Discussion for CICIDS 2017 Dataset	261
9.3	Results Analysis and Discussion for Multi-Class Datasets	268
9.3.1	Results Analysis and Discussion for NS-3 Multi-Class Dataset . . .	268
9.3.2	Results Analysis and Discussion for CICIDS 2017 Single Category Multi-Class Dataset	273
9.3.3	Results Analysis and Discussion for CICIDS 2017 Multi Category Multi-Class Dataset	277
9.4	Summary	283
10	Conclusion	287
10.1	Overview	287
10.2	Summary of Contributions	287
10.3	Limitations and Future Work	289
10.4	Conclusions and lessons learnt	294
	References	297

List of figures

1.1	Smart City Technologies	2
1.2	Sensors/Actuators functioning in IoT	3
1.3	IoT Applications in Smart City	3
1.4	Smart Home - Smart Grid Network	5
2.1	NIST Smart Grid Architecture	18
2.2	Smart Home Architecture and interaction with the Smart Grid	21
2.3	Trajectories Separation in Phase Space by δZ	43
4.1	Slow-Rate Packet Behaviour	75
4.2	ForChaosLR Multi-Layer IDS Architecture	81
4.3	ForChaosLR Multi-Layer IDS Architecture (multi-class)	82
4.4	Simple Exponential Smoothing Forecasting in legitimate traffic	83
4.5	Simple Exponential Smoothing Forecasting in DDoS attack	85
5.1	Smart Home-Smart Grid network simulation in NS-3	112
5.2	Arff file source code	119
6.1	Experiments Independent Variables Diagram for NS-3 smart home-smart grid dataset	124
6.2	30-second Window Size RMSE Error Rate for 3000 seconds for NS-3 smart home-smart grid dataset	135
6.3	40-second Window Size RMSE Error Rate for 3000 seconds for NS-3 smart home-smart grid dataset	136
6.4	60-second Window Size RMSE Error Rate for 3000 seconds for NS-3 smart home-smart grid dataset	136
6.5	ForChaos Summarised Results for NS-3 smart home-smart grid dataset . . .	156
6.6	LR second-layer Training Results 80/20 for NS-3 smart home-smart grid dataset	161

6.7	LR second-layer Training Results 10-Cross Validation for NS-3 smart home-smart grid dataset	161
6.8	ForChaos first-layer and ForChaosLR NS-3 Binary Summarised Dataset Results	164
6.9	ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (TP rates)	167
6.10	ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (FP rates)	168
6.11	ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (Precision rates)	169
6.12	ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (F-Score rates)	170
7.1	ISCXIDS2012 15th June Flooding Attack Packet Arrival Time/sec	181
7.2	ISCXIDS2012 15th June Flooding Attack Lyapunov Exponents Behaviour (no re-train)	182
7.3	ISCXIDS2012 15th June Flooding Attack Lyapunov Exponents Behaviour (after re-train)	182
7.4	ForChaos v.s. ForChaosLR TP and FP rates on ISCXIDS2012 Flooding Day (15th June)	183
7.5	ForChaos and ForChaosLR ISCXIDS2012 12th June Normal Behaviour Results	184
7.6	ISCXIDS2012 12th June Normal Behaviour Packet Arrival Time/sec	185
7.7	ISCXIDS2012 12th June Normal Behaviour Lyapunov Exponents Behaviour	186
7.8	ISCXIDS2012 12th June Normal Behaviour Lyapunov Exponents Behaviour with Re-training	187
7.9	ForChaos and ForChaosLR ISCXIDS2012 Results (total)	188
7.10	ISCXIDS2012 ForChaosLR and ML algorithms TP rates	192
7.11	ISCXIDS2012 ForChaosLR and ML algorithms FP rates	193
7.12	ISCXIDS2012 ForChaosLR and ML algorithms Precision rates	194
7.13	ISCXIDS2012 ForChaosLR and ML algorithms F-Score rates	195
7.14	ForChaos and ForChaosLR 2017 Results Flooding Day TP and FP rates	196
7.15	ForChaos and ForChaosLR 2017 Results Slow-Rate Day TP and FP rates	198
7.16	ForChaos and ForChaosLR CICIDS 2017 Results TP and FP rates (total)	199
7.17	CICIDS 2017 ForChaosLR and ML algorithms TP rates	200
7.18	CICIDS 2017 ForChaosLR and ML algorithms FP rates	201
7.19	CICIDS 2017 ForChaosLR and ML algorithms Precision rates	202

7.20	CICIDS 2017 ForChaosLR and ML algorithms F-Score rates	203
8.1	ForChaosLR and ML algorithms TP rates for NS-3 multi-class dataset . . .	235
8.2	ForChaosLR and ML algorithms TP-multi rates for NS-3 multi-class dataset	236
8.3	ForChaosLR and ML algorithms FP rates for NS-3 multi-class dataset . . .	237
8.4	ForChaosLR and ML algorithms Precision rates for NS-3 multi-class dataset	238
8.5	ForChaosLR and ML algorithms F-Score rates for NS-3 multi-class dataset	239
8.6	ForChaosLR and ML algorithms TP rates for CICSIDS2017 single category multi-class dataset	240
8.7	ForChaosLR and ML algorithms TP-multi rates for CICSIDS2017 single category multi-class dataset	241
8.8	ForChaosLR and ML algorithms FP rates for CICSIDS2017 single category multi-class dataset	242
8.9	ForChaosLR and ML algorithms Precision rates for CICSIDS2017 Sin- gleSlowRate dataset	243
8.10	ForChaosLR and ML algorithms F-Score rates for CICSIDS2017 SingleSlowRate dataset	244
8.11	ForChaosLR and ML algorithms TP rates for CICSIDS2017 multi category multi-class dataset	245
8.12	ForChaosLR and ML algorithms TP-multi rates for CICSIDS2017 multi category multi-class dataset	246
8.13	ForChaosLR and ML algorithms FP rates for CICSIDS2017 multi category multi-class dataset	247
8.14	ForChaosLR and ML algorithms Precision rates for CICSIDS2017 multi category multi-class dataset	248
8.15	ForChaosLR and ML algorithms F-Score rates for CICSIDS2017 multi category multi-class dataset	249

List of tables

2.1	IoT Examples Devices Specifications	26
2.2	DDoS Attacks	33
3.1	Related Studies	68
4.1	ForChaosLR Features Table	77
5.1	Summary of datasets	98
5.2	Smart Home-Smart Grid network protocols	113
5.3	NS-3 DDoS attacks traffic generation parameters	115
6.1	NS-3 smart home smart grid Scenarios Independent Variables	124
6.2	ForChaosLR multi-layer IDS Training Parameters for NS-3 smart home-smart grid dataset	129
6.3	ForChaos first-layer Training Scenarios for NS-3 smart home-smart grid dataset	132
6.4	ForChaos Experimental Results against Constant Attacks (2000 and 4000 seconds attack start time and $\alpha = 0.1$)	134
6.5	ForChaos scenarios for constant attacks (40 and 60 seconds window size and $\alpha = 0.1$)	138
6.6	ForChaos first-layer results against constant attacks - 40 and 60 seconds window size and $\alpha = 0.1$ results	138
6.7	ForChaos first-layer results against increasing attacks - 40 and 60 seconds window size and $\alpha = 0.1$	140
6.8	ForChaos first-layer results against increasing attacks - 40 and 60 seconds window size and $\alpha = 0.1$ results	141
6.9	ForChaos first-layer results against mixed attacks - 40 and 60 seconds window size and $\alpha = 0.1$ scenarios	144
6.10	ForChaos first-layer results against flash crowd events - 40 and 60 seconds window size and $\alpha = 0.1$ scenarios	146

6.11	ForChaos first-layer results against flash crowd events with 5 nodes - 40 and 60 seconds window size and $\alpha = 0.1$ results	148
6.12	ForChaos first-layer results against flash crowd events with 6 nodes - 40 seconds window size and $\alpha = 0.1$ results	149
6.13	ForChaos first-layer results against flash crowd events with 7 nodes - 40 seconds window size and $\alpha = 0.1$ results	150
6.14	ForChaos first-layer results against restart attacks - 40 seconds window size and $\alpha = 0.1$ scenarios	152
6.15	ForChaos first-layer results against restart attacks (2000 seconds attack start time) - 40 seconds window size and $\alpha = 0.1$ results	152
6.16	ForChaos Variables Optimised Values for NS-3 smart home-smart grid dataset	157
6.17	LR second-layer optimal variables NS-3 smart home-smart grid dataset . .	160
6.18	LR-second Layer Summarised Experimental Results for NS-3 smart home-smart grid dataset	164
6.19	Machine Learning algorithms Parameters	166
7.1	LR second-layer variables ISCXIDS 2012	178
7.2	LR second-layer variables CICIDS2017	178
7.3	ForChaosLR multi-layer CICIDS 2017 Application Layer DDoS Day Individual Results	198
7.4	ForChaosLR multi-layer Smart Home 2018 Results	208
7.5	ForChaosLR multi-layer Smart Home 2019 Results	209
8.1	ForChaosLR multi-layer variables multi-attack NS-3 Optimal Parameters .	218
8.2	ForChaosLR multi-layer variables multi-attack CICIDS2017 single-category multi-class Optimal Parameters	218
8.3	ForChaosLR multi-layer variables multi-attack CICIDS2017 multi-category multi-class Optimal Parameters	219
9.1	NS-3 Binary Training Set Hardware Requirements Metrics	253
9.2	NS-3 Binary Test Set Hardware Requirements Metrics	257
9.3	ISCXIDS 2012 Training Set Hardware Requirements Metrics	258
9.4	ISCXIDS 2012 Test Set Hardware Requirements Metrics	262
9.5	CICIDS 2017 Training Set Hardware Requirements Metrics	263
9.6	CICIDS 2017 Test Set Hardware Requirements Metric	267
9.7	NS-3 Multi-Class Training Set Hardware Requirements Metrics	268
9.8	NS-3 Multi-Class Test Set Hardware Requirements Metrics	273

9.9	CICIDS 2017 Multi-Class Single Category Training Set Hardware Requirements Metrics	274
9.10	CICIDS 2017 Multi-Class Single Category Test Set Hardware Requirements Metrics	278
9.11	CICIDS 2017 Multi-Class Multi-Category Training Set Hardware Requirements Metrics	279
9.12	CICIDS 2017 Multi-Class Multi-Category Test Set Hardware Requirements Metrics	283

Chapter 1

Introduction

This chapter serves as the introduction of the thesis. Firstly, a general overview is given followed by a motivation and specific research question and objectives. Then, a list with the thesis contributions is presented. Finally, the published work relevant to the thesis is included along with the thesis outline.

1.1 Problem Statement and Motivation

1.1.1 Smart City and IoT

For the last couple of decades, overpopulation has been a major issue for humanity. Overpopulation causes a number of problems that decrease a city's sustainability and effective functioning such as environmental pollution, emissions exhaustion, global warming, energy waste, health problems, public transport congestion and overcrowding. In detail, according to Economic and Social Research Council (ESRC), can significantly improve urban sustainability using information and communication technology (ICT) through multiple ways. Some examples include smart monitoring and management of parking spaces, quick and sustainable garbage collection, optimised usage of energy resources (e.g. water, electricity) and safe driving mechanisms [65].

Furthermore, the demand for water, electricity, gas and the need for smarter transportation, healthcare and safety solutions is more vital than ever. Hence, new smarter ways for living have become a necessity for humanity. As a result, the concept of “smart city” has emerged [60].

Smart city aims to improve the overall quality of life for its citizens, enhance sustainability and protect the environment. To succeed, it makes usage of intelligent technologies such as the cloud, blockchain, data analytics, artificial intelligence, machine learning, sensors/actuators

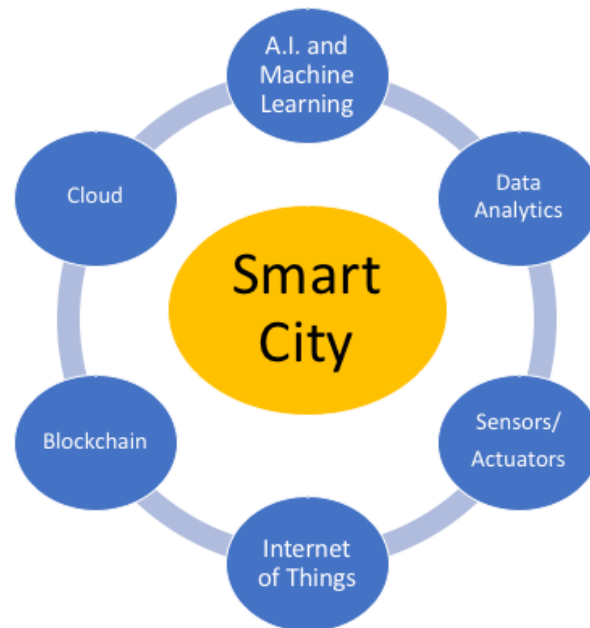


Fig. 1.1 Smart City Technologies

and the Internet of Things (IoT) as illustrated in Figure 1.1. This set of technologies can optimise the functioning of the city's core components and transform them in "smart entities" to match the citizens' needs.

It is generally believed that one of the most influential technologies towards the realisation of a smart city is the Internet of Things. IoT devices are estimated to reach 25 billion by 2025 [52]. IoT denotes the main link of the physical entities with electronic and mechanical machinery, software applications, sensors and actuators that are constantly connected to the Internet, collect data from the environment, exchange such data either with other intelligent devices located in the same network or in other networks and forward them to the relevant data centres and/or the cloud. Furthermore, IoT offers the ability for "things" to be monitored and/or controlled remotely as illustrated in Figure 1.2.

Therefore, with the integration of IoT, the efficiency of systems is increased, the accuracy is improved, the overall costs are significantly minimised and the probability of human error is substantially decreased. IoT integration examples include the intelligent transportation systems, smart water systems, smart waste systems, the smart grid, smart buildings, smart homes, smart agriculture systems, public safety systems (e.g. surveillance systems), smart government and agency services, smart healthcare systems (e.g. smart hospitals, smart remote medical services) and smart education systems as illustrated in Figure 1.3. IoT offers multiple benefits to the smart city and its citizens including: health building and bridges monitoring, sustainable energy, water and waste management, environmental and noise

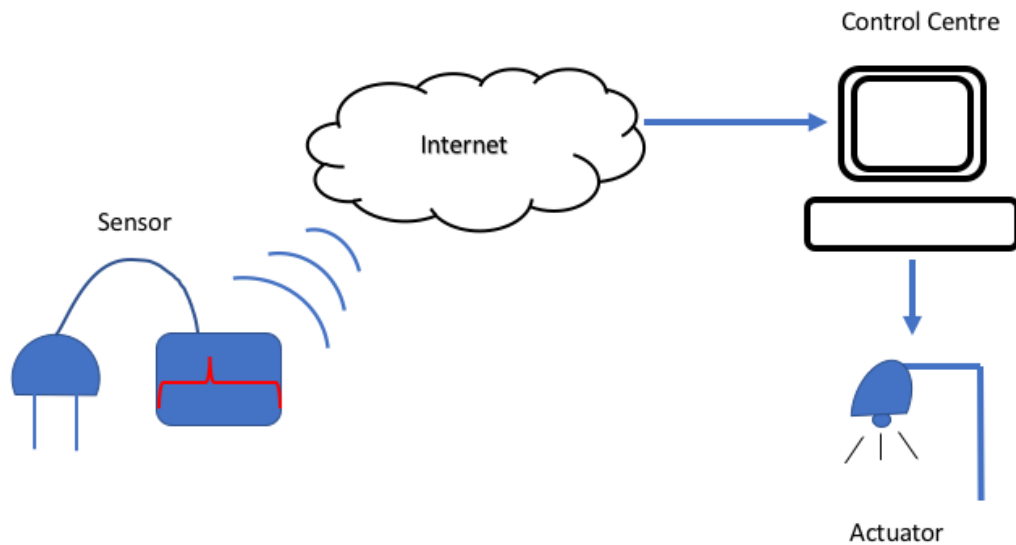


Fig. 1.2 Sensors/Actuators functioning in IoT

IoT applications in Smart City

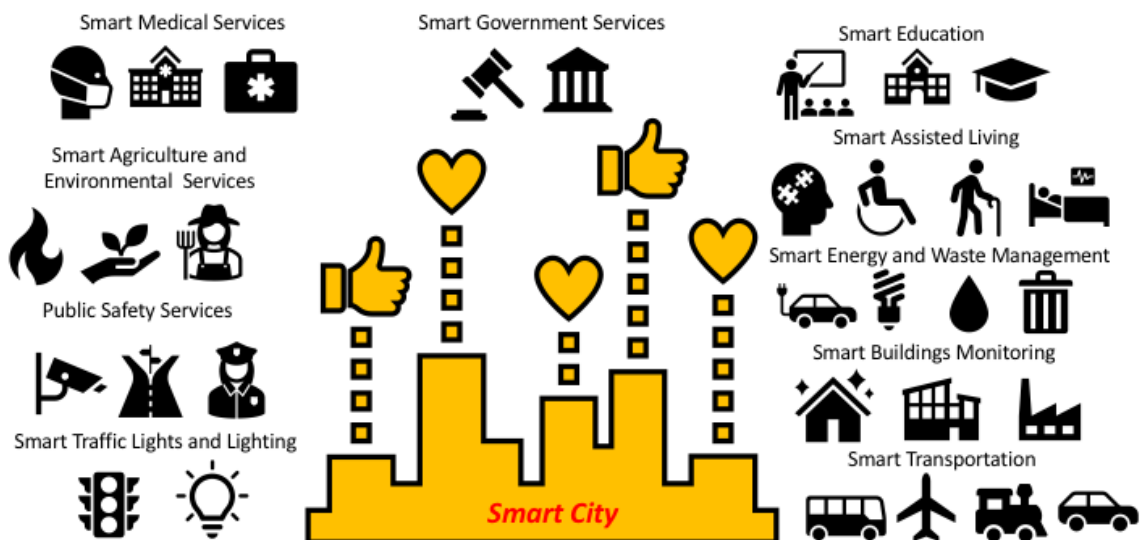


Fig. 1.3 IoT Applications in Smart City

monitoring, smart lighting, smart traffic lights and traffic congestion management, smart parking, remote health capabilities, public safety and intelligent transportation capabilities and assisted living.

1.1.2 Smart Home-Smart Grid network and its benefits

Perhaps, the most popular interconnection between two of the smart city networks is the smart home with the smart grid. As part of a smart city, the smart home-smart grid network assists residential owners in their everyday tasks and improve their quality of life. In more detail, with this duplex communication between the two networks, smart city citizens have now the opportunity to manage their electricity consumption in a more efficient and environmentally friendly way, decrease their monthly electricity bill, participate in energy saving up reward programmes, have their smart appliances and intelligent devices monitored and maintained remotely by relevant third-party services and exchange energy to the smart grid (provided they have energy distributed services installed in their premises).

Smart homes consist of a number of different devices, all deployed in a single network monitoring the environment, collecting and sharing important data and information with the owners and other smart IoT devices and external services through internal and external networks. The node responsible for this communication is the energy services interface (ESI). It acts as a bi-directional interface where information can be exchanged between the smart home and external domains. Furthermore, it protects internal energy resources from security failures and ensures secure internal communication between the devices deployed in the smart home. ESI's importance to the smart home and in the outside domains makes it an excellent target for cyberattacks.

The smart grid forms an alternative solution to the power grid. It is one of the most pioneering creations of our century. Smart grid is a modern, reliable, scalable and environmentally friendly infrastructure that keeps up to the latest requirements of the century and consists of the latest trends in technology as well [140]. The smart grid enhances the traditional electrical power grid's capabilities and also brings more benefits to the world. Some include the accurate real-time monitoring, optimised power flows, enabling of two-way communication between the users and the utilities and improvement of the procedures from an environmental point of view.

Due to the two-way communication, the smart home owners are now able to actively participate in the smart grid concept and adapt their consumption behaviour depending on their needs with the help of a smart meter installed in the premises. The smart meter is able to accurately record electricity consumption and report back to the AMI utility as well as report any outing issues that might rise. They can directly communicate with their utility provider. Their main objective is to assist their owners into their everyday tasks but also improve their overall quality of life.

Through the ESI, the smart home communicates with multiple facilities of the smart grid domain such as the advanced metering infrastructure (AMI) for meter reading, effective

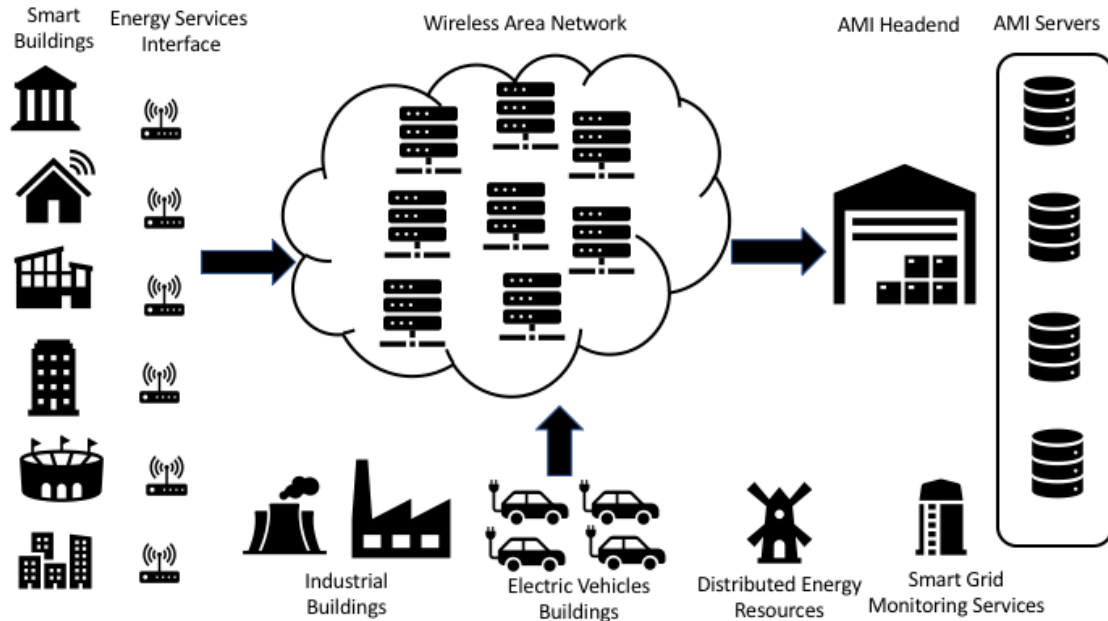


Fig. 1.4 Smart Home - Smart Grid Network

electricity usage and reporting issues to the relevant servers responsible for each procedure [172], [246], [87].

On the edge of the AMI network, the AMI headend accepts incoming data and forwards it to the relevant servers. The main relevant servers interacting with the consumers are Demand Response Management System (DRMS), Load Management System (LMS), Customer Information System (CIS), Outage Management System (OMS) and Data Management System (DMS). More information about their functioning is given in Chapter 5, Section 5.4.1.

Beyond residential smart homes, other smart buildings communicate with the AMI. Examples include large commercial and industrial premises, small commercial and industrial premises, distributed energy resources, individual plug-in electric vehicles and fleet of plug-in electrical vehicles (PEVs) [204]. Furthermore, the smart grid domain includes other sub-systems which monitor the normal functioning of the smart grid. In case of any issues, these systems report back to the AMI headend, transmitting important information regarding the issue so relevant actions are taken. A representation of this ecosystem is illustrated in Figure 1.4.

1.1.3 Smart Home-Smart Grid network Security Concerns

Based on the concepts described above, it is a fact that smart homes, other smart commercial and industrial buildings and smart grid monitoring networks constantly communicate with the AMI servers. Hence, there is tight connectivity between the multiple networks and entities through common network routes [73], [8], [89], [173], [53]. Due to this high inter-connectivity, it is possible to perform cross-network attacks from one network to the other. Specifically, from the smart home to the AMI services of the smart grid. This is the most likely to occur scenario, as a smart home is likely to have the least amount of security measures deployed. In addition, the heterogeneous nature and weak security measures (due to the devices being resource-constrained) make the smart home IoT devices and smart appliances attractive targets to adversaries. Therefore, the act of intrusion and compromise of such devices on the smart home side is relatively easy.

DDoS attacks in the Smart Home-Smart Grid network

Distributed denial of service (DDoS) attacks are in favour of being used to harm the smart home-smart grid network for numerous reasons. Firstly, a DDoS attack has a critical impact on the smart home-smart grid network, as its services are decreased or even become unavailable which as a result, can put human lives at risk.

Secondly, as already stated, the security countermeasure on the smart home side can be weak to non-existent. Hence, the fast and easy compromise of a large amount of devices and the formation of a botnet to conduct large-scale DDoS attacks is a high likelihood scenario. In the past, there were previous IoT botnets involved in DDoS attacks. Hence, IoT devices appear to be a popular choice for botnet masters. The most well-known and popular example of an IoT botnet is Mirai [79]. Mirai is a malware that infects and compromises IoT devices which in turn continuously scan the Internet for other IoT devices to infect further. Mirai compromises these devices by using a table of 60 common factory default usernames and passwords.

Mirai was responsible for some of the most major and high-in-impact IoT-based DDoS attacks. Examples included the DDoS attack on Brian Krebs' web site, an attack on French web host OVH, and the October 2016 Dyn cyberattack [10] which caused the disruption of major websites such as Twitter, Spotify, Netflix, GitHub, Amazon and Reddit with more than 100,000 of the compromised nodes being IoT devices. In Liberia, shortly after the Dyn incident, Mirai botnet again was used to shut down the entire Internet of the country [57] with a reported 500 Gbps throughput.

Other examples included Aidra, Bashlite and IRCTelnet [126]. Adversaries are after IoT devices since they are easily exploited, they are always online, security standards are not defined yet, rarely monitored and it is low in cost for an IoT device to be compromised. IoT botnets are mostly used for DDoS attacks. As a result, despite the benefits provided from the adoption of IoT into our lives, there are multiple security concerns that must be considered. Therefore, their deployment should be achieved with caution.

Application layer DDoS attacks issues and challenges

A popular type of DDoS attack is the application layer DDoS attack. Application layer DDoS attacks are increasingly popular among adversaries, due to having a greater impact on the target when compared to DDoS attacks in lower layers [92] [105]. In detail, the adversary requires less resources since the target's pool of connections accepted is not infinite and therefore easier to be overwhelmed faster. Furthermore, in such an attack, the target is most likely an individual host and not the network. Therefore, the attack is solely focused on the target chosen by the adversary. Finally, application layer DDoS attacks are undeniably stealthier due to their ability to mimic legitimate behaviour to a great extent [241], [92].

Application layer DDoS attacks can be broadly divided in two types: flooding and slow-rate. A flooding application DDoS attack aims to flood the target with a burst amount of legitimate requests which the server cannot refuse but to reply. As a result, it becomes unresponsive due to great amount of incoming requests. Application layer flooding DDoS attacks are harder to be detected than their equivalent ones in lower layers as they do not saturate the bandwidth and make the target unavailable by flooding it with malformed packets as it can happen in a TCP SYN flooding DDoS attack. In detail, in a SYN flooding DDoS attack the attackers transmit a burst amount of SYN packets without acknowledging the SYN/ACK packets sent by the server and thus not completing the 3-way handshake. In contrast, in a flooding DDoS attack on the application layer this is not the case as the TCP connection is established successfully and the requests sent have legitimate format and are compliant with the HTTP protocol rules. Furthermore, due to its "legitimacy", a flooding DDoS attack is often mistaken as flash crowd since both of them have the same behaviour: a sudden burst of legitimate requests entering a network in a short amount of time.

In contrast to flooding, slow-rate attacks are much stealthier, therefore they do not need high bandwidth to be conducted. Slow-rate application layer DDoS attack exploits a server's ability to wait for connections to be completed in a certain amount of time, in case the incoming connection is legitimately slow. As long as the client manages to send a subsequent packet to complete the request, the server is obliged to keep the connection open.

In web applications, a server has a finite pool of requests that can accept and reply with the relevant command. In cases which this pool is full, then the server (target in a cyberattack) is unable to respond to legitimate requests. In an attempt to avoid any security countermeasures that measure the sudden increase in traffic (too much noise), they exploit the vulnerability of some types of servers that are forced to wait for a request to be completed as long as the client (attacker in a cyberattack) keeps this request open by sending a subsequent packet up to a specific time interval. Essentially, the result is the same with a flooding DDoS that generates a lot of noise; the server's requests' pool is overwhelmed thus, making it not available to requests from legitimate users/devices.

In more detail, the "time to wait for a request" is a real-world functionality by servers which was the main inspiration for constructing such attacks. Every server is bound to wait for a specific amount of time before it rejects the connection. However, as long as the client (attacker in a cyberattack) keeps on sending a subsequent packet, the "time to wait for a request" value is reset. This is a frequent legitimate scenario as not all devices can complete a request in one go due to different reasons such as slow Internet connection, older devices capabilities and so on. To avoid any security countermeasures that measure the sudden increase in traffic (too much noise), they exploit the vulnerability of some types of servers that are forced to wait for a request to be completed as long as the client (attacker in a cyberattack) keeps this request open by sending a subsequent packet up to a specific time interval. There are well-known tools that conduct different types of slow-rate attacks such as Slowloris, SlowHTTPTest, RUDY that conduct different variations of slow-rate attacks [202], [178], [205], [167].

Based on this vulnerability, the slow-rate attack opens a great number of connections and initiates requests that never completes them, resulting in the target becoming unresponsive. Once again an application layer slow-rate DDoS exposes the fact that it successfully and legitimately completes a TCP connection and sends a valid request to the server that "is seemingly incomplete from the first packet".

This could be a valid scenario especially in cases which the internet connection is "slow". Hence, we observe that application layer flooding DDoS attacks are harder to be detected due to their "seemingly legitimate behaviour" and similarity to flash crowd events. Slow-rate attacks are harder to be detected due to their similarity with slow connections trying to complete a request legitimately.

Numerous studies were conducted to detect application layer DDoS attacks such as [243] and [244]. Although they achieve remarkable results, they aim to detect application layer DDoS attacks which are conducted against web servers using user-web behaviour approaches. Specifically they monitor human-behaviour web characteristics such as the number of clicks,

number of links clicked on, time spent on a web-page and so on. Unfortunately, this is not applicable in our situation since the application layer traffic the smart home-smart grid generates is not necessarily user-centric.

Countermeasures against DDoS attacks and research question

In DDoS detection research literature, machine learning algorithms are in strong favour since they are accurate in detecting unknown malicious behaviour with a minimum rate of false positive (FP) rates. However, they have high computational complexity which results in high needs of memory and CPU power. In addition, such algorithms require a relatively long training time and a large in size training set. Thus, they can cause computational burden in network systems which consist of resource-constrained devices such as the smart home-smart grid during training time. In addition, since they are high in complexity and need more resources, they are not flexible when needed to be retrained. Therefore, the security solutions deployed in the smart home-smart grid network should be lightweight so it does not cause additional resource burden.

We believe that the interaction between the smart home and the smart grid can potentially result into the frequent re-training of an IDS. Specifically, according to numerous papers such as [61], [59], [117], [114] the smart grid has the capability to protect itself from overloading by letting home-owners participate in demand reduction, triggered by peak load or system contingencies in addition to bill saving programmes.

As a result, the overall traffic generated is likely to go through sudden and unexpected changes (mainly in the former case but also in the latter one to a lesser degree). This can potentially cause the IDS in place to produce a considerable number of false alarms since the traffic load is likely to be significantly reduced.

Extending our previous argument, we also believe that more use-cases of sudden change in traffic can occur, resulting from everyday human-based scenarios happening inside the smart home. Examples include: the installation of a set of new additional smart home appliances (e.g. new sets of smart kitchen appliances, new set of lighting installations, outdoors/garden smart sensors etc.) or the arrival of guests bringing their own devices to connect to the network (e.g. group online gaming, social gatherings etc.).

Based on the two arguments provided, we believe that re-training is a concept worth of considering due to the interaction between the two networks, the use cases inside the smart homes and the number of devices and their behaviour not being fixed and distinct.

The detection techniques used in lightweight approaches construct a baseline of normal behaviour and immediately identify anything that crosses this baseline as malicious. Indeed,

lightweight approaches are less of a computational burden to IoT devices however, according to the literature they report a rather high false positive rate (FP Rate).

Therefore, there is need for a multi-layer detection algorithm that minimises the FP rate and still be able to detect malicious behaviour but at the same time not overstressing to a device as well as forwarding the anomalous traffic to investigate it in further [108], [40].

This thesis examines the appropriateness of a multi-layer detection system for application layer DDoS attacks in IoT networks. Specifically, the thesis examines the possibility of multi-layer artificial intelligence system to effectively detect application layer flooding and slow-rate DDoS attacks in IoT networks (such as the smart home-smart grid network) with the following research question:

"Could a lightweight multi-layer IDS accurately detect application layer flooding and slow-rate DDoS attacks in the smart home-smart grid network and correctly differentiate it from legitimate traffic?"

1.2 Objectives

Our research is focused on designing implementing and evaluating an intrusion detection system (IDS) for the smart home-smart grid network, called ForChaosLR multi-layer IDS. ForChaosLR should be lightweight so it does not negatively impact any of the resource-constrained and IoT devices present in the smart home-smart grid network. ForChaosLR is focused on detecting application layer DDoS attacks, specifically flooding and slow-rate. It should also be effective in accurately differentiating these stealthy and seemingly legitimate attacks from normal network behaviour. Hence, ForChaosLR consists of a multi-layer approach to improve detection results, classify legitimate network traffic correctly and maintain its lightweight nature. Proceeding, we outline the main objectives of this thesis.

1. Research on the interaction and functionality of critical nodes and communication protocols in the smart home-smart grid network.
2. Research how application layer DDoS flooding and slow-rate attacks are conducted in the smart home-smart grid network and what their impact is.
3. Research the relevant literature regarding lightweight and multi-layer intrusion detection using A.I. and machine learning techniques.
4. Based on the knowledge acquired from Objectives 1 and 2, simulate the smart home-smart grid network normal and malicious behaviour and construct an accurate and balanced dataset in NS-3 network simulator.

5. Based on the knowledge acquired from Objectives 3 and 4, design a lightweight multi-layer IDS for the smart home-smart grid network to classify unknown traffic as malicious or legitimate.
6. Implement the lightweight multi-layer IDS for the smart home-smart grid network to accept incoming unknown traffic and classify it as normal or malicious generated from the simulated smart home-smart grid network based on the model designed.
7. Evaluate the ForChaosLR multi-layer IDS through a series of extended experiments to test the proposed solution's overall accuracy, error rate and complexity both using the NS-3 dataset and other real-world datasets to assess both ForChaosLR multi-layer IDS's accuracy and the accuracy of the NS-3 dataset simulation.
8. Deploy ForChaosLR multi-layer IDS in a resource-constrained device (Raspberry Pi 3 model B+) to evaluate on whether ForChaosLR overstresses this device (in terms of memory usage, CPU power and training/testing time required).
9. Expand ForChaosLR multi-layer IDS to perform multi-attack classification by implementing the relevant code accordingly and perform a series of relevant evaluations to assess its accuracy.
10. Implement popular machine learning algorithms and evaluate them on the same datasets and the resource-constrained device (Raspberry Pi 3 model B+) used for ForChaosLR multi-layer IDS to compare their results.

1.3 Contributions

The main contributions provided from our thesis include the following:

1. ***Comprehensive and in-depth research on the state of the art regarding intrusion detection.***

An in-depth investigation and analysis was conducted in intrusion detection. In detail, the investigation focused on IDS systems focused on multiple topics that were of interest for our research. These topics include DDoS detection in all TCP/IP (transmission control protocol/internet protocol) layers, application layer DDoS detection, lightweight IDS systems in resource constrained networks, IDS systems using forecasting techniques, chaos theory, regression and other machine learning and A.I.

approaches and IDS systems following a multi-layer approach. Finally, after summarising the key findings related to these studies, we identified the literature gaps and limitations when applied to our research study.

2. *Simulation of the smart home-smart grid network and construction of a smart home-smart grid dataset*

Based on the knowledge regarding the smart home-smart grid network acquired, we proceeded by simulating the smart home-smart grid network based on the relevant and latest standards, specifications and protocols used. Next, we simulated flooding and slow-rate DDoS attacks in the smart home-smart grid network as well as flash crowd events using metrics from real-world incidents occurred to make the attacks as realistic as possible. Finally, we generated and collected enough data to construct a balanced dataset to use for the ForChaosLR multi-layer IDS's evaluation.

3. *Design, implementation and evaluation of ForChaosLR multi-layer IDS*

Proceeding, we proposed ForChaosLR, a lightweight multi-layer approach for the smart home-smart grid aiming to detect application layer flooding and slow-rate DDoS attacks. After presenting ForChaosLR multi-layer IDS using the relevant mathematical concepts, we described how ForChaosLR multi-layer IDS should operate and outlined the set of experiments our ForChaosLR multi-layer IDS solution should be evaluated.

ForChaosLR multi-layer IDS is focused in the smart home-smart grid network. To evaluate our proposed solution, we implemented a smart home-smart grid network simulation using NS-3 network simulator [75].

To construct ForChaos first-layer we used Java programming language. To train and construct the LR second-layer and other machine learning algorithms, for comparison reasons, we used the Weka toolbox [144]. Through our experiments, we introduced different scenarios of application layer DDoS flooding and slow-rate attacks. All the relevant information are described in Chapter 5, Section 5.4.5 and 5.2.1. Furthermore, to gain more in depth results we made usage of other real-world datasets that consist of application layer flooding and slow-rate DDoS attacks. The relevant information about these datasets are described in Chapter 5, Section 5.2.2.

After evaluating ForChaosLR multi-layer IDS's performance in differentiating malicious traffic from legitimate, regardless of the attack type, we proceeded with evaluating ForChaosLR multi-layer IDS's performance in correctly categorising malicious traffic to the relevant specific attack type. We also compared its performance to the set of machine learning algorithms used in the previous set of experiments. Finally, we

deployed ForChaosLR multi-layer IDS and other machine learning algorithms on Raspberry Pi 3 model B+ to evaluate their hardware requirements and assess their suitability for low-resource constrained devices.

4. *Dissemination of Results and Future Directions*

Proceeding, we presented the results of our experimental studies. In detail, we evaluated both detection layers of the ForChaosLR multi-Layer IDS as well as the IDS system as a whole. We also compared ForChaosLR to popular machine learning algorithms which were also trained using our created NS-3 datasets as well as the other real-world datasets chosen. Finally, we discussed the results obtained from the hardware requirements on a Raspberry Pi 3 model B+ of ForChaosLR and the machine learning algorithms chose. Through them, we effectively supported that a lightweight IDS approach can accurately detect application layer flooding and slow-rate DDoS attacks, correctly categorise them in their respective attack type and not overstress a resource-constrained device that is deployed on. We also effectively supported that a lightweight multi-layer approaches can improve overall results when it comes to reducing FP rates than one-layer approaches.

1.4 Published Work

- A. Procopiou and N. Komninos, (2015). Current and Future Threats Framework in Smart Grid Domain. Paper presented at the 5th Annual IEEE International Conference on CYBER Technology in Automation Control and Intelligence Systems, 8-06-2015 - 12-06-2015, Shenyang, China.
- A. Procopiou, N. Komninos, and C. Douligieris, (2019). ForChaos: Real Time Application DDoS detection using Forecasting and Chaos Theory in Smart Home IoT Network. *Wireless Communications and Mobile Computing*, 2019, 8469410.. doi: 10.1155/2019/8469410
- A. Procopiou, A. and N. Komninos(2019). Bio/Nature-inspired algorithms in A.I. for malicious activity detection. In: El-Alfy, E-S. M., Elroweissy, M., Fulp, E. W. and Mazurczyk, W. (Eds.), *Nature-Inspired Cyber Security and Resiliency: Fundamentals, techniques and applications*. . IET. ISBN 978-1-78561-638-9
- A. Procopiou, T.M Chen, “Security and Privacy Issues, Challenges and Solutions in IoT networks for the Smart Cities” in *Internet of Things: A hardware development*

perspective (Part of the Prospects in Networking and Communications – P-NetCom Series) by CRC Press of Taylor and Francis Group, 2021 (accepted).

- A. Procopiou, T.M. Chen, “Explainable Artificial Intelligence in machine/deep learning for malicious activity detection in intelligent transportation systems for smart cities” in Explainable Artificial Intelligence for Smart Cities by CRC Press, Taylor & Francis Group, 2021.
- A. Procopiou, T.M. Chen, “Malicious Activity in IoT Networks: A nature-inspired approach” Advances in Nature-inspired Cyber Security and Resilience, Springer, 2021.

1.5 Thesis Outline

In this section, the overall structure of the thesis is outlined. In Chapter 2, the relevant background knowledge regarding the smart home-smart grid network and its security issues is provided. Proceeding, relevant background knowledge on intrusion detection systems is provided as well as popular detection techniques used such as machine learning, forecasting, chaos theory and regression analysis.

In Chapter 3, the related studies are presented regarding IDS and assessed. In the end, a critique is made based on the recent studies previously presented and the literature gaps are highlighted.

In Chapter 4, our ForChaosLR multi-layer IDS proposed system is introduced. In this chapter, we outlined its architecture, overall functioning, what mathematical concepts are adopted and how the two layers interact with each other. Proceeding, we described the relevant training and evaluation procedure for each layer separately. Finally, we provide the relevant pseudocode for both binary and multi-class classification. In the latter, only LR-second layer is affected, specifically its training procedure since more classes are involved in training.

In Chapter 5, the experimental setup and tools and programming languages for the implementation of the ForChaosLR multi-layer IDS and the smart home-smart grid network simulation are presented. Firstly, all the relevant information regarding the datasets used in the experimental evaluation chapters are presented. Proceeding, an in-depth explanation of the smart home-smart grid network simulation using the NS-3 simulator [75] is provided. This description includes the smart home-smart grid network and relevant IoT application layer protocols implementation, traffic generation and DDoS attacks simulation. Next, the important parts of ForChaosLR implementation are provided and explained, focusing more

on the forecasting and Lyapunov exponents implementation in Java and the Weka toolbox [144].

In Chapter 6, the first series of evaluation for ForChaosLR multi-layer IDS is conducted using the NS-3 dataset constructed. Firstly, the training procedure takes place for both layers of ForChaosLR multi-layer IDS, since they follow a different training approach, so their optimal parameters are found. Then, additional scenarios from the NS-3 dataset (Chapter 5, Section 5.2.1) evaluate both layers' accuracy in further. Finally, machine learning algorithms are evaluated against the NS-3 dataset and their results are compared with ForChaosLR multi-layer IDS.

In Chapter 7, ForChaosLR multi-layer IDS is evaluated using the real world datasets, presented in Chapter 5, Section 5.2.2. Firstly, the training procedure takes place for both layers of ForChaosLR multi-layer IDS, since they follow a different training approach, so their optimal parameters are found. Then, ForChaosLR multi-layer is evaluated through the respective testing sets for every real-world dataset used. Finally, machine learning algorithms are evaluated against the real-world datasets and their results are compared with ForChaosLR multi-layer IDS.

In Chapter 8, the series of multi-attack evaluation experiments for ForChaosLR multi-layer IDS are conducted using both the NS-3 and real world datasets. Firstly, the training procedure takes place for both layers of ForChaosLR multi-layer IDS, since they follow a different training approach, so their optimal parameters are identified. Then, ForChaosLR multi-layer is evaluated through the respective testing sets for every real-world dataset used and for the NS-3 dataset. Finally, machine learning algorithms are evaluated on the NS-3 and real-world datasets and their results are compared with ForChaosLR multi-layer IDS.

In Chapter 9, the first series of experiments on Raspberry Pi 3 model B+ for ForChaosLR multi-layer IDS are conducted using both the NS-3 and real world datasets, defined and explained in Chapter 5, Sections 5.2.1, 5.2.2 and 5.2.3. Memory usage, CPU power and execution time metrics are monitored for every dataset, for both training and testing phases separately. Finally, machine learning algorithms are evaluated against the NS-3 and real-world datasets and their results are compared with ForChaosLR multi-layer IDS.

Proceeding, the Raspberry Pi 3 model B+ resources evaluation for ForChaosLR multi-layer IDS is conducted using both the NS-3 and real world datasets. Memory usage, CPU power and execution time metrics are monitored for every dataset, for both training and testing phases separately. Finally, machine learning algorithms are evaluated against the NS-3 and real-world datasets and their results are compared with ForChaosLR multi-layer IDS.

Finally in Chapter 10, the thesis is concluded. In detail, the contributions made are summarised. Then, the limitations and future directions our research study can take are outlined for an effective extension the system. Lastly, a set of lessons learnt in the thesis is presented.

Chapter 2

Background Knowledge

2.1 Overview

In this chapter, a review of the relevant background knowledge and mathematical concepts related to the thesis is presented. In detail, all the necessary knowledge regarding smart home-smart grid network is given, starting from the network architecture and protocols used. Then, we proceed to the cybersecurity aspects of IoT networks, such as the smart home-smart grid network, defining its vulnerabilities, threats and relevant cyberattacks, focusing on DDoS attacks.

Proceeding, we provide relevant background knowledge on intrusion detection systems (IDS). Specifically, we describe their relevant concepts that have to do with their architecture, data collection, decision approaches and detection mechanisms. Since the detection engine is the most important component in an IDS, we focused our research on the most notable detection techniques presented in the literature, with some examples being forecasting techniques, chaos theory, regression analysis and machine learning approaches.

2.2 Smart Home-Smart Grid Architecture

To explain the smart home-smart grid network architecture we must firstly review each network architecture on their own and then proceed to their combination.

2.2.1 Smart Grid Architecture

Smart grid can be defined as a conceptual network of networks that encompasses different domains that communicate with each other to provide different services. This concept was effectively captured by NIST [86]. The NIST smart grid proposed architecture contains seven

different network domains. These are bulk generation, transmission, distribution, operation, market, service provider and customer. Each domain is briefly explained in detail below. A diagram of the NIST architecture is also presented in Figure 2.1 [86].

Conceptual Model

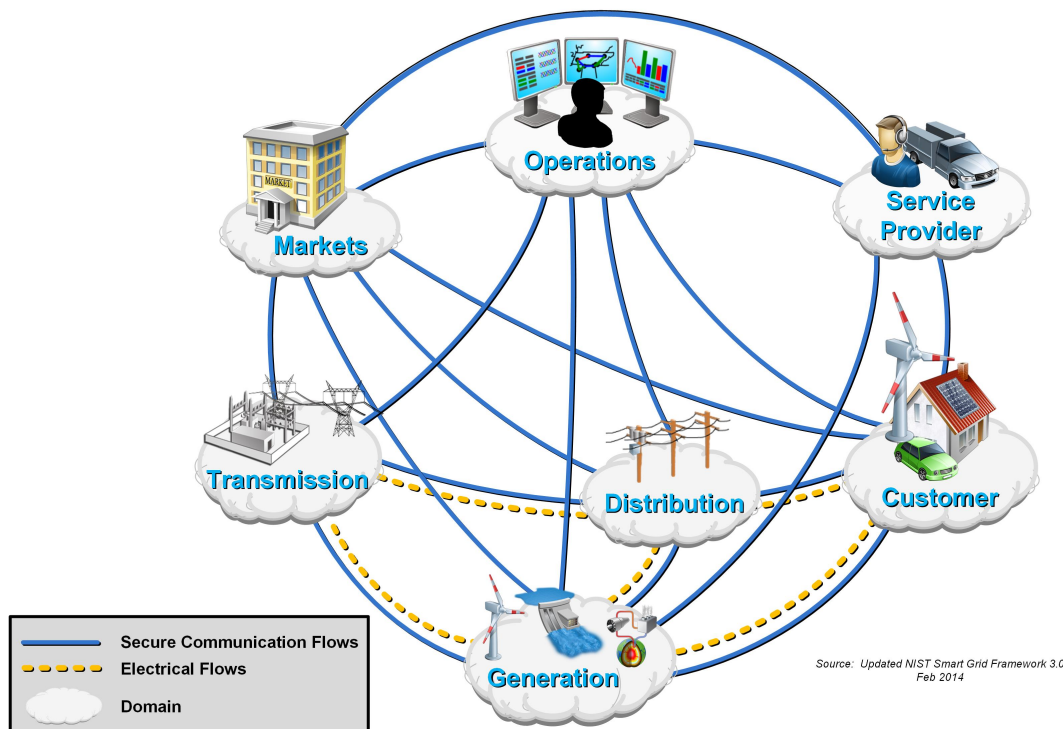


Fig. 2.1 NIST Smart Grid Architecture [86]

Bulk Generation domain deals with the overall generation of electricity. Various types of electrical equipment are deployed such as remote terminal units (RTUs), programmable logical controllers (PLCs), equipment monitors and fault recorders. This domain also communicates and interacts with the market and operation domains. It transmits forwards information such as generation capacity and scarcity [86].

Transmission domain acts as a beacon between the bulk generation and the distribution domain. It transfers electricity forwarded from the bulk generation to the distribution domain through substations and transmission lines [86].

Distribution communicates with the customer domain so it can deliver electricity to the end users based on their demands and electricity availability. Also, the distribution domain communicates with distributed energy resources, plug-in electrical vehicles (PEVs) and advanced metering infrastructures (AMI) [86].

Operation domain ensures the efficient and optimal functioning of transmission and distribution domains. It makes usage of the field and wide area networks (FAN and WAN) deployed in both transmission and distribution domains. Through them, valuable information are collected about the grid's state using supervisory control and data acquisition (SCADA) systems [86].

Market domain ensures and maintains the balance between the electricity supply and demand. This is ensured by the proper communication with the bulk generators and the distributed energy resources [86].

Service Provider domain provides electricity to the different types of customers. Their operations include billing and management of customer accounts. The domain receives all relevant metering information from the operations and provides smart services to the customer domain [86].

Customer domain denotes the direct consumers that make usage of electricity. In some cases, the individuals on this domain can store and generate their own electricity. The customer's side can directly and actively benefit from its interaction with the smart grid. Previously, this was not possible as the electrical grid did not provide any sort of two-way communication [86].

The Customer domain includes all the residential premises however there are other types that belong to this domain. These are large commercial and industrial premises, small commercial and industrial premises, individual plug-in electric vehicles as well as fleet of PEVs according to [204].

This domain communicates with the distribution, market, service provider and operation domains. Using the two-way communication provided between the smart home and the smart grid, the customers can monitor and manage their electricity consumption. With the introduction of the smart grid, demand/response programmes were introduced to reduce the electricity consumption when the market price is high or when the smart grid is overloaded due to demand peaks. Hence, customers can save up energy, cost and get rewarded for their environmentally friendly behaviour from the service provided. Furthermore, they can produce their own energy and sell it to the market if they have the relevant distributed energy resources installed in their premises. In addition, they have direct access to their smart metering and billing services.

The residential premises in the Customer domain are also known as smart home networks. Smart home consists a part of the Internet of Things vision, a network of smart devices and appliances that belong to different applications such as home automation, lighting control, climate control and entertainment and safety systems. Smart homes can positively influence our personal lives, promote sustainability, significantly reduce our daily energy consumption

and improve our quality of life. Some examples include controlling home access, managing home appliances, measuring home conditions, remote access and control of the overall smart home devices, saving energy and being friendly to the environment, [166], [118], [132], [96].

2.2.2 Smart Home Architecture

The gateway providing this two-way communication for the smart home is the ESI (energy services interface). Through ESI, electricity usage is measured, recorded and transmitted towards the advanced metering infrastructure (AMI). The smart meter can also act as a gateway for these services. ESI allows for remote connection and disconnection of services either from maintenance staff or from the customers themselves. ESI also receives pricing and demand/response signals and notifies the customers. ESI mainly interacts with the AMI (advanced metering infrastructure) for the mentioned services.

The smart meter is also another important entity of the smart home. It measures, records and monitors energy usage of all the smart appliances and devices of the smart home. It is also responsible for reporting outage issues and enables automating provisioning and maintenance. Furthermore, it is used for measuring and recording important meter events regarding the power quality management and possible damage or malicious handling of the meter.

The different smart and IoT entities in the smart home network can be divided into smart appliances, safety and security systems, entertainment systems, healthcare systems and smart sensors. Examples of smart appliances are smart air-conditioners, wash-machines, ovens, refrigerators, fridges, laundry machines. Examples of safety systems are smoke, fire and flood detectors. Examples of security systems are security cameras and smart door-locks. Examples of entertainment systems are smart TVs, media players, laptops and wireless speakers. Examples of healthcare systems are smart wristbands, portable ECGs and pulse oximeters. Examples of smart sensors are light bulbs and strips, air quality sensors and humidity sensors. In addition, smart cars and distributed energy resources can be deployed in the smart home network.

2.2.3 Smart Home-Smart Grid Architecture

Our proposed architecture of the smart home-smart grid network is illustrated in Figure 2.2 and described as follows: ESI acts as an information management gateway between the customers and third-party services, such as energy services providers, gas/water utilities and medical/security equipment, monitoring besides sending and receiving demand/response and

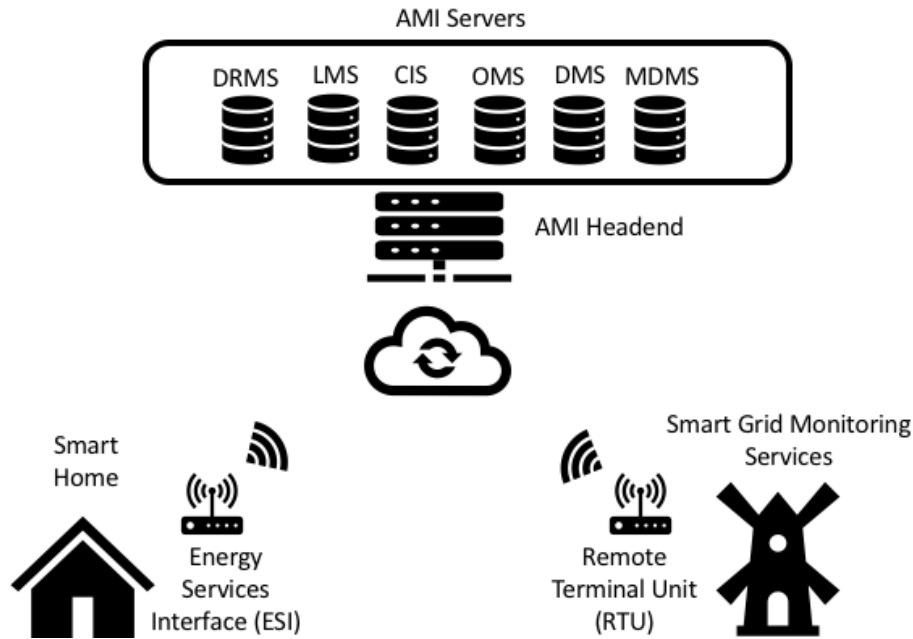


Fig. 2.2 Smart Home Architecture and interaction with the Smart Grid

load shedding signals from the utilities and also assisting the customer into enlisting new smart appliances.

These servers are located in the AMI infrastructure of the smart grid. ESI transmits all relevant information to the AMI headend which upon validation is forwarded to servers of the smart grid. Meter data management system (MDMS) is responsible for analysing the consumption data and eventually preparing the billing. Load management system (LMS) sends load-shedding signals and Demand response management System (DRMS) sends and receives demand/response signals. Customer information system (CIS) is the intermediate layer between customers and services, helping the former manage their smart appliances and their energy consumption programs. Outage management system (OMS) is responsible for receiving outage management reports and notifying the relevant teams to repair the outages. Finally, device management system (DMS) is responsible for the overall management of devices, including remote-control and maintenance.

Furthermore, except the AMI services in the smart grid, there are also the self-healing services of the smart grid, electricity monitoring services, substation monitoring services, SCADA systems and distributed renewable energy stations. All of these services communicate with the AMI Headend to report about this part of the smart grid's reliability and overall functioning. Devices in this domain of the smart grid include smart electricity meters, phasor measurement units, relays, intelligent electronic devices, digital fault detectors, renewable energy generators and sensors [249]. In addition, other networks from the customer domain

frequently communicate with the AMI services such as small/large commercial and industrial premises as well as networks of PEVs [204].

2.2.4 Smart Home-Smart Grid Network Communications Protocols

Smart Home Communications Protocols

- ***Hypertext Transfer Protocol(HTTP)***

Hypertext transfer protocol (HTTP) is considered a request-response protocol between a client and a server [131]. The client constructs and sends a request to the server. The server receives the request and proceeds with checking and validating the request. Upon validation, the server responds accordingly.

The HTTP protocol consists of different request methods. These methods help the server choose the most suitable response to send based on the request received. The most popular requests are GET and POST. GET consists of requesting a specific resource from the server while POST consists of sending a specific set of data to the server. The semantics of the HTTP request-response mechanism structure are simple and straightforward. The request message contains the following:

- A request line.
- The relevant header lines.
- An empty line.
- A message body. The message body is optional for the GET request and mandatory for the POST request.

All request lines must always have a "/r/n" at the end to indicate the end of the message.

- ***Constrained Application Protocol (CoAP)***

CoAP is a synchronous request/response application layer protocol between a client and a server. It was designed by IETF (Internet engineering task force) in 2014, for resource constrained devices such as wireless sensor network (WSN) nodes [41]. CoAP is designed to support multicast and consists of very low overhead and simplicity. These features are important and essential for IoT and machine-to-machine(M2M) devices. CoAP was inspired from the HTTP headers and contains four types, namely: GET, POST, PUT, and DELETE. CoAP is located on top of the UDP (user datagram protocol) protocol so the overhead caused by the TCP (transmission control protocol) can be avoided and the implementation remains lightweight. However, reliability

should be ensured. For that reason, two bits are used in the header of each packet, indicating the type of message and the required quality of service level. The message type options are the following: Confirmable, Non-Confirmable, Acknowledgment and Reset which are briefly explained below.

1. Confirmable: A request type of message containing an acknowledgement (ACK). It can be sent either synchronously or asynchronously.
2. Non-Confirmable: A message that does not need to be acknowledged.
3. Acknowledgment: A message confirming that the message was delivered successfully.
4. Reset: A message confirming that was not delivered.

- ***Message Queue Telemetry Transport (MQTT)***

MQTT is another lightweight protocol for resource-constrained devices such as M2M and IoT proposed in 2014. It was developed by IBM [22] as an asynchronous publish/subscribe protocol. In contrast to the CoAP, it is located at the top of the TCP protocol. The publish/subscribe functionality requires a message broker between the different clients. The broker is responsible for distributing messages to clients depending on the message's topic and their interest. Every client can be a publisher who can transmit a message broker about a specific topic. Every client can also be a subscriber and receive updates on messages they are interested in from a broker. Despite MQTT's lightweight nature, reliability still needs to be ensured. MQTT achieves that by providing the option of three QoS levels:

1. Fire and forget: Send a message once without acknowledgement required for it.
2. Delivered at least once: Send a message at least once and acknowledgement is required for it.
3. Delivered exactly once: Use a four-way handshake mechanism to ensure the message is delivered exactly once.

- ***Extensible Messaging and Presence Protocol (XMPP)***

XMPP protocol was standardised by IETF in 2011, mainly for chatting and message exchanging purposes [197]. Similarly to MQTT, XMPP is located at the top of the TCP layer. It provides asynchronous publish/subscribe and synchronous request/response messaging systems. These functionalities are ideal for M2M, IoT and other types of resource-constrained devices. On the other hand, XMPP uses XML (extensible

markup language) in its messages which unfortunately creates additional overhead due to the unnecessary tags contained. Furthermore, XMPP needs XML parsing thus, requiring additional computational resources. As a result, the power consumption is increased which can prove a critical issue for resource-constrained devices, especially the ones that consist of a battery as their only energy resource. The XMPP architecture is similar to the email architecture as there is no central master server and anyone can run XMPP on their server. In addition, functionality can be customised on top of the protocol such as chat-rooms, file sharing, VoIP and cloud services. In contrast to the previous IoT protocols described, XMPP provide no reliability mechanisms for assuring of the messages delivery by default. Hence, such functionality has to be developed on top of the XMPP, once again causing additional computation overhead.

- ***Advanced Message Queuing Protocol (AMQP)***

AMQP was proposed in 2006 and forms an open standard application layer protocols used for message-oriented middleware [227]. Similarly to XMPP and MQTT, it is an asynchronous publish/subscribe communication protocol and is also placed at the top of the TCP protocol. AMQP provides numerous different features, including message orientation, queuing, routing (including point-to-point and publish-and-subscribe). Once again, since AMQP is a lightweight protocol reliability is ensured by a specific header which can be any of the following types:

1. At most once: the message is transmitted once whether of it being delivered or not.
2. At least once: the message is transmitted at least once, if not more.
3. Exactly once: the message is transmitted only once.

- ***Websockets***

Websockets protocol was developed as part of the HTML5 and standardised in 2011 by IETF [74]. It provides a full duplex communication between a client and a server and it is placed on top of the TCP protocol, similarly to MQTT, XMPP and AMQP. Despite Websockets protocol being full-duplex, it reduces the communication overhead since it is neither a request/response nor a publish/subscribe protocol. Similarly to the HTTP, the client commences a handshake with the server to establish a websocket session. Once this session is established, the HTTP headers can be removed and the client can exchange messages with the server using a full-duplex asynchronous connection. Hence, both HTTP and Websockets can be processed using the same port.

Smart Grid Communication Protocols

- ***Distributed Network Protocol (DNP3)***

DNP3 forms a set of communication protocols mainly used in electric/water utilities, companies and organisations [78]. It is mainly responsible to ensure the proper and successful communication and data exchange between SCADA systems, RTUs (remote terminal units) and the various intelligent electronic devices placed.

- ***IEC 61850***

IEC 61850 is primarily used as a communications protocol in electrical substations. It is responsible for ensuring the proper communication and data exchange between intelligent electronic devices and automation services [149]. The main goal is to achieve interoperability between devices developed by different vendors. The level of details includes protocols and the data types used in communication.

- ***IEC 62056 or DLMS/COSEM***

IEC 62056 is responsible for electricity meter data exchange. In practice, it is used by smart meters to send information to the smart grid regarding a smart building's electricity consumption behaviour [228].

2.2.5 Smart Home-Smart Grid Network Devices Specifications

As described, IoT devices encompass resource-constraints with regards to memory, storage and processing power. Hence, IETF proposed a set of standards specified for the IoT devices. This set of standards classifies them in the following categories [38]:

1. Class 0: Devices which contain the lowest resources (<10 kB of RAM and <100 kB Flash) in terms of memory and processing. They are very constrained with regards to secure communication. Therefore, they are most likely to be connected to proxies, gateways and servers for their effective internet communication. These devices have their minimal network stack occupying most of their resources. Hence, they are incapable of encompassing any additional security and application layers.
2. Class 1: Devices which contain medium-level resources (10 kB of RAM and 100 kB Flash). These devices consist of a low power IoT stack. This stack contains popular protocols such as UDP, CoAP and DTLS. However, they are unable to employ a full protocol stack which makes usage of heavyweight protocols such as HTTP and TLS without the involvement of a gateway.

Table 2.1 IoT Examples Devices Specifications

<i>Device</i>	<i>Chipset</i>	<i>Core Freq.</i>	<i>RAM</i>	<i>Flash Memory</i>	<i>Class No.</i>
iPhone	A7x Quad-core	1.7Ghz	2GB	128GB	undefined
Nest Thermostat	ARM Cortex-A	800Mhz	512MB	2GB	undefined
Nest Smoke-Detector v1	ARM Cortex-M4	100Mhz	128KB	512KB	2
Nest Smoke Detector v2	ARM Cortex-M0	48Mhz	16KB	128KB	1
NETGEAR Router	BCM4709A	1.0Ghz	256MB	128MB	undefined
Samsung Smart TV	ARM Exonys SoC	1.3Ghz	1GB	N/A	undefined
Samsung SmartCam	GM812x SoC	540Mhz	N/A	64GB	undefined
Elster REX2 Smart Meter	71M6531F SoC	10Mhz	4KB	256KB	0
Philips Hue Light bulb	TI CC2530 SoC	32Mhz	8KB	256KB	0
Fitbit Smart Wrist Band	ARM Cortex-M3	32Mhz	16KB	128KB	1
Sensor Devices	Microcontroller	4 -32Mhz	4 -16KB	16 - 128KB	0-1

3. Class 2: Devices which contain more resources compared to the previous two classes (50 kB of RAM and 250 kB Flash). These devices are able to support most of the protocol stacks. However, these protocols should be lightweight and consider energy-efficient solutions as well as less bandwidth consumption.
4. Classes beyond 0, 1, 2: Devices which consist of higher capabilities compared to the previous classes devices. They are able to make usage of most of the available communication protocols. IETF has yet to define any additional classes. However, it is stated that even devices beyond classes 0, 1 and 2 are still constrained by limited energy supply. For that reason, major companies such Huawei , ARM, and Google focused on IoT software [94].

In Table 2.1 we summarise examples of IoT smart home devices as described in [133], with their device specifications.

2.3 Smart Home-Smart Grid Network Security

2.3.1 IoT devices Security Issues and Vulnerabilities in the Smart Home-Smart Grid Network

Due to the overall varied nature of IoT devices compared to traditional computer hosts, additional security issues should be considered [12], [209], [90], [141], [95], [24].

- **Resource Constraints:** Most of the IoT devices and appliances, consist of low-power processors, reduced-size memories and low in size storage. As a result, they encompass

decreased computation and storage capabilities. For that reason, existing complex and high in resources cryptographic algorithms, authentication protocols and intrusion detection systems are unlikely to properly function when deployed on them. As a result, studies were conducted that perform the training phase separately. Specifically, the training should be conducted in a data centre/cloud and then the constructed model should be deployed on the IoT device. Alternative solutions include the training procedure to be split on different conceptual layers of the IoT network in an attempt to reduce the resources needed for training. Some notable examples include [168], [112], [181].

- ***Interconnection and Interaction of Networks:*** The smart home-smart grid network generates a large amount of data (big data) due to the very large number of devices deployed and the different set of communications occurring between them. This set of data is used for the extraction of useful patterns and trends for better analysis and decision-making. However, this interconnection and constant interaction can potentially cause incorrect-authorisation to users issues. Due to lack of sophisticated security countermeasures, improper software updates and unchanged factory settings (e.g. default passwords), IoT devices can be easily compromised. The interconnection of a large number of IoT devices can cause a single compromised device to be the entry point of an attack which can lead to the compromise of multiple devices and their catholic misbehaviour.
- ***Energy Constraints:*** Numerous IoT devices operate solely using their battery power resources. As a result, these devices contain limited energy which is significantly decreased when the computation requirements are moderate to high in computational resources.
- ***Physical Access:*** IoT devices in a smart home-smart grid network may need to be scattered along large geographical areas, especially on the smart grid side. Therefore, the scattered devices might not be constantly monitored by human personnel and thus, an easy target for physical attacks by adversaries.
- ***Legacy Devices:*** In the smart home-smart grid network, the smart grid side is likely to consist of legacy devices that are integrated along with new intelligent electronic and IoT devices. These legacy devices can introduce new security attack vectors due to their outdated nature.
- ***Open APIs:*** Most of the APIs for the IoT are open to promote active participation in the development of different applications and possible improvement on the functionality.

This is likely to cause issues in the smart home-smart grid network as potential software vulnerabilities could be discovered and exploited to the adversary's advantage.

- **Mesh Networks:** IoT sensors are particularly popular and effective when it comes to the monitoring of an environment (e.g. air quality sensors deployed to monitor the atmosphere quality). The sensors can exchange data directly with each other through a mesh network. However, the absence of a central authority monitoring the communications might lead to undetected malicious behaviour.
- **Mobility of Nodes:** Some of the IoT devices in the smart home-smart grid network are likely to be mobile, mainly on the smart home side. Thus, disruptions between them are highly likely to occur. This makes the efficient detection of compromised and misbehaving devices difficult.
- **No clear communication between the different teams:** For the effective functioning of the smart home-smart grid network, numerous professionals from different backgrounds have to cooperate. As a result, there is a risk for inefficient communication between the different teams of professionals since there are differences in knowledge and experience of the individuals. This can lead to disorganised and incorrect decisions that might affect the services of the smart home-smart grid network.
- **Unsophisticated security mechanisms:** Due to the IoT devices' low capabilities in memory, processing power and energy resources it is a challenge to deploy sophisticated authentication mechanisms. These ineffective and insufficient authentication mechanisms (e.g. weak/default passwords) are likely to be exploited by adversaries. Another aspect of this vulnerability consists of weak encryption mechanisms which can result in encrypted data to be cracked by an adversary. Furthermore, IoT devices are likely to have more open ports than expected due to them running various services. An adversary is likely to find these relevant open ports after conducting scanning the network with its hosts and exploit the vulnerable devices.
- **Insufficient patch management:** As any other operating system, IoT operating systems should be patched frequently and appropriately. In that way, attack vectors used by adversaries can be minimised. However, the relevant manufacturers are unlikely to release security patches or do not provide any functionality for automatic patch-updates.
- **Weak programming practices:** Various IoT firmware have been released with known and popular vulnerabilities such as backdoors, root access, and lack of secure socket

layer (SSL) presence. Exploiting these vulnerabilities, an adversary is likely to conduct various attacks such as buffer overflows data illegal modification or unauthorised access.

2.3.2 IoT Attacks Taxonomy

In IoT networks, such as the smart home-smart grid network, there are numerous cyberattacks that can be conducted against networks and/or individual assets. These attacks can threaten one or more of the main cybersecurity principles : confidentiality, integrity and availability. Proceeding, we outline the different attacks, focusing on how these can be conducted in IoT networks [154], [67], [91] and [62] focusing on D/DoS attacks. D/DoS attacks in IoT networks can actually negatively affect the operational capabilities of the device or the entire IoT network as the authors in [193] effectively stated.

Physical Attacks

As already described, IoT devices can be directly physically reachable in the smart home-smart grid network. Hence, their hardware components, such as RFID (radio frequency identification) readers, sensors/actuators and RFID tags are vulnerable to numerous types of hardware attacks [91]. Proceeding, we briefly describe the most popular and notable physical-based attacks in IoT networks.

- **Node Replication Attack:** An adversary deploys a physical object in the network to act maliciously (e.g. corruption/misdirection of packets, data leakage) and cause issues to network normal functioning [70].
- **RF Interference on RFID DoS attack:** This type of DoS attack consists of the adversary sending a burst amount of noise signals over radio frequency signals to negatively impact the RFID communication between the different nodes [136].
- **Hardware Trojan:** The adversary maliciously alters the integrated circuit of the device so they can gain access to the circuit's firmware and data [29].
- **Power Outage Attack:** The adversary can deliberately turn off or reduce the power of IoT devices [91].
- **Jamming DoS:** Using a jammer, the adversary can disrupt the wireless communication between IoT nodes, ultimately causing a DoS attack and making the target node unavailable [62]. Jamming can either be constant or deceptive and occurs at the

physical layer of the TCP/IP protocol stack. In the former the adversary sends a burst amount of radio signals over a short period of time. In the latter, the adversary sends a false radio signals to cause the target becoming unavailable. Proceeding, we briefly described some examples of jamming D/DoS attacks in IoT networks.

- RF jamming, where the attacker sends RF signals so they can interfere with the communication between different devices. As a result, the devices fail to communicate with each other [62].
 - WSN node jamming where the attacker aims to disrupt the wireless communication between nodes in a wireless sensor network by actively jamming the wireless channel [62].
 - Wide-band and pulse denial attack, where the attacker blocks the radio frequency on the IoT network completely and the communication is fully lost [62].
- Physical Damage: An adversary takes advantage of IoT devices being deployed over large geographical areas and not constantly monitored to physically harm them and make them unavailable [62].
 - Node Tampering: Since IoT devices are physically accessible, an adversary can directly reach them and tamper them in multiple ways including extracting cryptographic keys, altering the firmware or the operating system or modifying the circuit [98].
 - Malicious Node Injection: Aiming to acquire unauthorised access, the adversary physically inserts a malicious node in an IoT network. Through it, they can gain access to other IoT nodes, inject false data, drops or re-direct packets elsewhere [91].
 - Malicious Code Injection: The adversary inserts physically malicious code inside an IoT node so they can compromise it [62].
 - Side Channel Attack: The adversary aims to break IoT devices' security countermeasures (e.g. encryption) by analysing different types of side channel information generated by them [165].
 - Sleep Deprivation Attack: In this attack, the adversary aims to exhaust the IoT devices' power resources and eventually shutting them down [62].
 - Tag Cloning: Many IoT devices have tags attached on them for monitoring and geo-location tracking reasons. However, an adversary can capture these tags from a short distance and replicate them. Proceeding, using the replicated fake tags they can bypass

relevant tag readers [165]. A popular example in RFID-enabled IoT networks denotes the adversary copying wrong data from an existing RFID tag to a new RFID tag [17].

- **Social Engineering:** As the authors in [62] correctly described, adversaries can physically modify an IoT device to obtain sensitive data using social engineering techniques to deceive humans in the process. In the context of physically tampering, abusing or vandalising an IoT device they do not have physical access, social engineering techniques can assist in the adversary in coming to close proximity to the target IoT device without raising suspicions.

Network and Protocol-Based Attacks

These attacks focus on exploiting the IoT network functioning or the IoT protocols deployed.

- **Traffic Analysis:** The adversary intercepts network traffic and analyse messages exchanged between the different IoT nodes to obtain valuable information [62].
- **Spoofing (also known as Impersonation):** The adversary disguises themselves as a legitimate and trusted source in the network. Spoofing is mainly used to get access to confidential information, spread malware, bypass access controls mechanisms, or re-direct traffic in the context of a DoS attack. Popular examples include: ARP (address resolution protocol), MAC (medium access control), DNS (domain name server) and IP (internet protocol) spoofing attacks. In IoT wireless communications, popular examples include RFID and Bluetooth low energy spoofing attacks. In the former, the adversary spoofs the RFID signals and proceeds with gaining the information transmitted from the RFID tag [136], [128]. In the latter, a popular vulnerability of the Bluetooth low energy is exploited, which consists of the beacon being broadcasted publicly. Hence, the adversary can obtain the beacon's UUID, impersonate the beacon and bypass the rules to falsely verify its identity [91].
- **Eavesdropping (also known as Sniffing):** This attack denotes the interception of a, usually wireless, communication between two nodes by the adversary. In RFID-enabled IoT networks, all communications between the tags and readers are conducted wirelessly and without any encryption mechanisms. Therefore, any communications between the tag and reader can be easily intercepted by an adversary and private information can be leaked [91]. Another example includes the NFC wireless communications. NFC (near-field communication) protocol's lack of security mechanisms and close proximity between the nodes makes it subject to eavesdropping attacks [91].

- **Man in the middle (MITM):** In this attack, the adversary is placed in the middle of a communication between two specific nodes, pretending to be the intended node the legitimate ones try to communicate with. The adversary relays and possibly alters the data it intercepts [17]. In IoT networks, this attack can easily be conducted on RFID systems during a communication between a reader and tags. The adversary can intercept and modify the data exchanged over the communication channel [128]. MITM can also occur on NFC communications [91].
- **Unauthorised Access:** The adversary falsely gains permission to access a system, data or communications through privilege escalation. In IoT networks, this is a popular attack example in RFID systems, where the adversary takes advantage of the lack of authentication mechanisms and poses themselves as legitimate users to manipulate data exchanged [17], [6].
- **Replay Attack:** The adversary aims to re-send previous sent packets or delay their transmission. This can be used in the context of an attempt to impersonate a legitimate node and gain unauthorised access through privilege escalation. In tag systems, the attacker could intercept and illegally obtain the transmitted signal between the reader and the tag so they can use it at a later time [128].
- **Illegal Modification Attack:** The adversary aims to disrupt the communication between two nodes by altering the content transmitted [44].
- **(Distributed)/Denial of Service (D/DoS):** A D/DoS describes a set of different types of attacks aiming to make a target (individual host or the network itself) unavailable. A DoS can be distributed. A DoS consists of only one attacker conducting the attack while a DDoS consists of multiple attackers conducting it. D/DoS attacks can be volumetric or vulnerability exploitation-based.

A volumetric DDoS attack consists of an attacker sending a massive volume of traffic in an attempt to flood the target machine and make it unable to accept legitimate requests from normal users. In a vulnerability-exploitation DDoS, an attacker is taking advantage of a protocol or communication-based weaknesses to make the target unavailable to legitimate requests from normal users.

A D/DoS attack can also occur in any of the TCP/IP Layers. Proceeding, we categorise D/DoS attacks based on their common behaviour and briefly describe them. D/DoS attacks that use jamming as their primary source of malicious behaviour were already described earlier.

Table 2.2 DDoS Attacks

<i>DDoS Category</i>	<i>DDoS Description</i>	<i>DDoS Examples</i>
<i>Routing Attacks</i>	Disruption of the normal communication between IoT nodes by impersonation or modification/false routing information.	Sinkhole, Sybil ARP Poissn., Blackhole, Greyhole, Wormhole
<i>Exploitation Attacks</i>	Disruption the normal communication between IoT nodes by exploiting protocol vulnerabilities.	Land, Ping of Death, Smurf, Fraggle, Teardrop, Low-Rate, SSL DoS, Slow-Rate(Slow-send and Slow-Read)
<i>Flooding Attacks</i>	Disruption of the normal communication between the IoT nodes by flooding a specific target or the network itself.	UDP Flood, TCP SYN Flood, ICMP Flood, GET/POST Flood, HULK, Goldeneye

Proceeding, we explain the most popular DDoS attacks relevant to IoT networks and a summary of them is presented in Table 2.2.

- DDoS Routing Attacks: In this set of D/DoS attacks the adversary aims to disrupt the normal communication between the IoT nodes in the network by impersonating legitimate nodes, modifying or sending false routing information. As a result, packets can be dropped, re-directed or selectively forwarded. Proceeding, we briefly describe the most popular routing-based D/DoS attacks across the TCP/IP stack layers.
 - * Sinkhole attack: The adversary compromises one of the nodes of the actual IoT or WSN network and proceeds with sending fake routing information to the other nodes, falsely claiming to know the minimum distance path to the base station. The other nodes send the malicious node traffic which it either alters or drops [62].
 - * Sybil attack: The adversary illegally compromises the identities of one or more nodes and falsely behaves as them. One popular example is the in WSN networks where this attack occurs to negatively impact the voting mechanism installed [62].
 - * ARP Poisoning (also known as ARP Spoofing): In the media access layer (MAC), the attacker performs MAC address spoofing so it can send ARP messages to the IoT network. Hence, messages are redirected to them instead of the legitimate user. Therefore, the attacker can cause a DoS attack by dropping the packets. Another example of an ARP poisoning denotes

the attacker redirecting multiple IP addresses' traffic to a single host, thus making it unavailable [80].

- * **Blackhole:** In the network layer, a compromised node that is supposed to forward packets received drops them [180].
 - * **Greyhole:** This attack is similar to blackhole attack but the packet drop is conducted selectively to avoid detection, once again in the network layer [15].
 - * **Wormhole:** In the network layer, the attacker receives packets at one location in the IoT network, "tunnels" them to another location and replays them there. Once a wormhole is established, malicious nodes can use it to drop certain data or control packets [15].
- **DDoS Exploitation-based Attacks:** In this set of D/DoS attacks the adversary aims to disrupt the normal communication between the IoT nodes in the network by exploiting protocol vulnerabilities. Proceeding, we briefly describe the most popular exploitation-based D/DoS attacks across the TCP/IP stack layers.
- * **Land:** In the transport layer, the adversary sends a spoofed TCP SYN packet which contains the victim's IP address as both source and destination. This causes the victim to send replies to itself over and over again, resulting into it becoming unavailable [161].
 - * **Ping of Death (PoD):** In the network layer, the adversary sends an either malformed or oversized packet using the ping command, thus making the target unavailable [161].
 - * **Smurf:** In the network layer, this D/DoS attack can either be a simple or distributed DoS. The adversary sends a large number of ICMP echo requests to the IP broadcast address (by IP spoofing the victim's IP address), thus making the target unavailable [161].
 - * **Fraggle:** In the transport layer, this attack is similar to Smurf attack but instead of ICMP echo packets, UDP echo packets are sent to the target [161].
 - * **Teardrop:** In the network layer, the adversary sends mingled IP fragments, which are either overlapping or with very large in size payloads to the target server in an attempt to make it unavailable [161].
 - * **Low-Rate:** In the transport layer, the adversary initiates a TCP connection but transmits a partial TCP SYN packet. By exploiting the server's time to wait for a connection to be completed, the adversary transmits small subsequent packets just before the server drops the connection so it can remain open and

available. Proceeding, more connections are initiated with partial requests, but never completed. As a result, the target becomes unavailable [161].

- * **SSL DoS:** In secure sockets layer (SSL), the adversary exploits the SSL handshake authentication mechanism, thus making the target unavailable by sending malformed packets or by exploiting the functions of the SSL encryption key negotiation procedure [161].
- * **Slow-Rate :** In the application layer, these attacks consist of transmitting small amounts of traffic to the target by exploiting specific protocol vulnerabilities. We proceed by briefly describing the most popular types of such attacks [92].
 - **Slow-Send:** The adversary successfully completes a TCP handshake and proceeds with sending a partial application layer request. The adversary takes advantage of the server's time waiting for a request to be completed and strategically sends a small subsequent packet to maintain the request open and valid. Proceeding, more and more connections are initiated and the target ends up becoming unavailable. The adversary can choose to partially send the header or the body. The slow header version can use either a GET or POST request and can be executed with the popular tools Slowloris and SlowHTTPTest [92]. Slow body transmits the header successfully but the body is sent at a very slow pace. Popular tools for the slow body attack include RUDY, however the presence of forms is required, and SlowHTTPTest [92].
 - **Slow-Read:** The adversary successfully sends a request to the server which responds back accordingly. However, the adversary deliberately slowly consumes the response. As time goes by, more requests are maintained, thus making the target unavailable. Slow-read attack is implemented using the SlowHTTPTest tool [92].
- **DDoS Flooding Attacks:** In this set of D/DoS attacks the adversary aims to disrupt the normal communication between the IoT nodes in the network by flooding a specific target or the network. Proceeding, we briefly describe the most popular flooding DDoS attacks across the TCP/IP stack layers.
 - * **UDP Flood:** In the transport layer, the adversary transmits a burst amount of UDP packets, randomly and at different ports, towards the target, thus making it unavailable [161].

- * TCP SYN Flood: The adversary transmits a burst amount of TCP SYN packets to the target, thus making it unavailable [161].
- * ICMP (Ping) Flooding: In the network layer, the adversary transmits a burst amount of Internet control message protocol (ICMP) "echo request" (ping) packets to the target, thus making it unavailable [161].
- * Application layer GET Flooding: The adversary transmits a burst amount of legitimate identical GET requests, thus making the target unavailable [92].
- * Application layer POST Flooding: In the application layer, the adversary transmits a burst amount of legitimate identical POST requests, thus making the target unavailable [92].
- * HULK (HTTP Unbearable Load King) Flooding: In the application layer, the adversary transmits a burst amount of legitimate application layer packets to the target. However, instead of the requests being identical, they consist of random and unique patterns of data to evade detection and increase the load in servers [92].
- * Goldeneye Flooding: This is another flooding attack in the application layer, where the packets contain randomised payloads, header and request initialisation times [92].

Software Attacks

In this set of attacks, the adversary aims to conduct a attack using malicious software or by exploiting the existing software's vulnerabilities.

- Malware: In this attack, the adversary aims to alter the IoT device's behaviour through its infection with malicious code. Examples of malware include viruses, worms, Trojan horses, spyware and ransomware. For IoT devices specifically, the authors in [91] argued that RFID tags (which are frequently used in IoT devices with tags) can potentially be used as a medium for successful virus spreading.
- Malicious Script: Similarly to malware, the attacker injects malicious script on the IoT device so they can gain unauthorised access through privilege escalation [91].
- Bug Exploitation: In this attack the adversary exploits software vulnerabilities to bypass authentication mechanisms and gain unauthorised access through privilege escalation [91].

- **Buffer Overflow:** A buffer overflow consists of a programme unintentionally overrunning the buffer's while writing the input data. As a result, adjacent memory locations are overwritten. This misbehaviour can be maliciously conducted when an attacker injects a large amount of data that ends up overwriting part of the actual application. These adjacent memory locations are most likely to include executable code which as a result cause various issues such as access errors, incorrect results, programme crashes, authentication bypassing and reduce availability to legitimate users [62].

2.4 Intrusion Detection Systems

Intrusion detection systems (IDS) denote one of the most popular and effective security countermeasures against the attacks described. An IDS should be intelligent and sophisticated in its functioning. The main components of an IDS include the data collection, detection and alert components [223], [109]. Below, we describe each component:

Data Collection: In this component, relevant behaviour (e.g. network traffic, host behaviour) is collected. The collection can be conducted from one or multiple sources. Some examples of collected data include user and system data, network routing and data traffic from the network and log files. Based on a pre-defined set of features, which are useful for differentiating legitimate from malicious behaviour, network traffic or host behaviour is analysed. The analysis includes calculating a corresponding value for each of the selected features. Next, this set of values is forwarded to the detection component.

Detection Engine: The detection engine denotes the most important part of the IDS. It receives the values for the defined set of features and decides on whether there is an intrusion or not through a series of calculations. To achieve that, an algorithmic model is constructed during a training phase. Multiple algorithms can be used including machine learning (support vector machines (SVM), random forest, K-nearest neighbours (KNN), Bayesian methods, clustering), reinforcement learning (e.g Monte Carlo, Q-learning), deep learning methods (e.g. deep belief, recurrent, convolutional and long-short-term-memory neural networks), forecasting methods (linear/quadratic exponential smoothing forecasting) and regression methods (linear, multiple, multi-variate and logistic regression). Most of the research aims to improve the detection rate of these algorithms and lower the false positive rate by tuning the relationship between the algorithm itself and the features that are used for data collection.

Alert: This component generates an alert if there is an intrusion. This component might provide additional information regarding the type of the attack, hosts/services affected and personal information leakage. In addition, IDS systems may contain mitigation/prevention functionality.

Intrusion detection systems can be grouped into categories according to multiple criteria, including their data type collection, detection technique, architecture and decision making type. Below a brief summary of the most popular examples from each category [37].

2.4.1 IDS Taxonomy

Data Collection Taxonomy

- Network-based intrusion detection system (NIDS): This IDS aims to detect intrusions and malicious behaviour with regards to network-based threats. It achieves that by collecting incoming/outcoming network traffic.
- Host based intrusion detection system (HIDS): This IDS aims to detect malicious behaviour with regards to individual hosts' behaviour. All collected data used for analysis includes behaviour from individual hosts.
- Hybrid intrusion detection system: This IDS combines the two types of IDS systems defined above as it monitors both network traffic and individual hosts' behaviour.

Detection Technique Taxonomy

- Signature-based: Aims to detect malicious behaviour by relying on attack signatures which were already seen. Examples include traditional anti-virus and anti-malware systems. This technique is ideal for already seen attacks but fails for zero-day attacks.
- Anomaly-Based: Aims to detect malicious behaviour by collecting relevant data about the system's behaviour and comparing it with what is considered normal. Through the usage of mathematics, probabilities and statistics the system decides whether the deviation of observed behaviour from the expected is due to an attack or a legitimate anomaly in the system.

Decision Technique Taxonomy

- Collaborative decision making: Multiple IDS agents deployed collaborate in deciding on whether there is an intrusion.
- Independent decision making: Each IDS agent solely decides on whether there is an intrusion based on the events surrounding them.

Architecture Taxonomy

- **Centralized IDS:** A centralized IDS monitors all relevant behaviour (network traffic and/or host data) and detects intrusions.
- **Stand-alone IDS:** An IDS runs on each node to be guarded independently. Every decision is solely based on the information collected at its own node.
- **Distributed and cooperative IDS:** Each node runs an IDS agent which takes part in the intrusion detection and response. If a node detects an intrusion with weak evidence, then it can initiate a cooperative global detection procedure. If a node detects an intrusion locally with enough evidence to justify the intrusion, then it can generate an alert.
- **Hierarchical IDS:** This architecture is mainly proposed for clustering network infrastructures. Cluster-head IDS agents monitor their relevant member nodes inside the respective cluster, but also take part in global intrusion detection decisions.
- **Mobile Agent based IDS:** Each mobile agent is responsible for a specific task of the IDS on a selected node. The detection part is handled in a cooperative way from these selected nodes. After a certain time period or after a specific task is done, agents may move to other nodes for effective monitoring.

2.5 IDS Detection Techniques

2.5.1 Time-Series Forecasting Approaches

Simple Exponential Smoothing Forecasting

Simple exponential smoothing forecasting was firstly proposed by Robert Goodell Brown in 1956 [35] and then expanded by Charles C. Holt in 1957 [102] in an attempt to smooth time-series data. Simple exponential smoothing forecasting is also known as exponential weighted moving average (EWMA). Simple exponential smoothing forecasting calculates predictions for constant-level time series which do not demonstrate any trend or seasonality. Simple exponential smoothing forecasting is useful for data smoothing in signal processing and as low-pass filters to remove high frequency noise. Simple exponential smoothing forecasting assigns weights to past data that are used in the calculation of a prediction. The weights are decreased exponentially as the data is getting older. Simple exponential smoothing forecasting can be described mathematically as follows:

In an observed time series which consists of observations $A_1, A_2, A_3, \dots$ for an event, the simple exponential smoothing forecasting formula F_t is defined as the next forecast at time t . A_{t-1} is the previous actual observation, F_{t-1} is the previous forecast and α is defined as the smoothing constant. Smoothing constant α can take the values $0 \leq \alpha \leq 1$. The choice of α value plays an important role in having a balance between the forecast and actual observation. By having α leading towards 1, more weight is given to the actual observation. If α is leading towards 0, more weight is given to the past observations (which are used to construct the prediction value). The formula of simple exponential smoothing forecasting is shown in Equation 2.1.

$$F_t = \alpha \cdot A_{t-1} + (1 - \alpha) \cdot F_{t-1} \quad (2.1)$$

Double Exponential Smoothing

A considerable weakness of simple exponential smoothing forecasting is that it underperforms in cases which there is a trend in observed data [192]. A trend is defined as a decrease or increase of the average value during time. Hence, double exponential smoothing forecasting (also known as Holt's linear trend model) was proposed [113]. Double exponential smoothing forecasting is considered an extension of simple exponential smoothing forecasting as it introduces slope. Slope aims to improve the predictions quality together with the α constant.

Slope b ranges from $0 \leq b \leq 1$. In double exponential smoothing we symbolise the raw data as x_t , starting from $t = 0$. We symbolise the smooth data as s_t at time t and b_t the trend estimation at time t . The final forecast is symbolised at F_t . For $t = 1$, we define the smooth value s in Equation 2.2 and the trend value b in Equation 2.3. For $t > 1$, we define the smooth value s in Equation 2.4 and the trend value b in Equation 2.5. Finally, the forecast is calculated through the Equation 2.6 at time $t + m$.

$$s_1 = x_1, t = 1 \quad (2.2)$$

$$b_1 = x_1 - x_0, t = 1 \quad (2.3)$$

$$s_t = \alpha \cdot x_t + (1 - \alpha) \cdot (s_{t-1} + b_{t-1}), t > 1; s_1 = x_1, t = 1 \quad (2.4)$$

$$b_t = \beta \cdot (s_t - s_{t-1}) + (1 - \beta) \cdot b_{t-1}, t > 1; b_1 = x_1 - x_0, t = 1 \quad (2.5)$$

$$F_t = s_t + mb_t \quad (2.6)$$

2.5.2 Chaos Theory

2.5.3 Introduction

Chaos or (deterministic chaos) was firstly discovered and presented by Edward Lorenz in 1972 who defined it as [142]:

“Chaos: when the present determines the future, but the approximate present does not approximately determine the future. ”.

Chaos theory is a sub-area of mathematics that studies “chaos”. Chaos concerns dynamic deterministic systems which can, in principle, be predicted. However, the behaviour of such a systems might appear random and unpredictable after some time [66], [63]. However, despite its “randomness”, it actually contains underlying patterns, interconnectedness, constant feedback loops, repetition, self-similarities, fractals and self-organisation and deterministic laws between the states of it which can justify its unexpected and significant change [77].

2.5.4 Properties of chaotic systems

According to Robert L. Devaney, for a system to be classified as chaotic it must satisfy the three following properties [66]:

1. It should be sensitive to initial conditions.
2. It should be topologically transitive.
3. It should have dense periodic orbits.

Under certain circumstances, the second and third properties actually imply the first one [66]. This makes the first property the most important. Hence, we proceed by describing the first property in more detail.

Sensitivity to initial conditions was also firstly described by Lorenz in 1972 [142]. In his groundbreaking paper, he supports that seemingly sudden and unexpected significant changes in a system’s behaviour or not as random as thought. There have been small changes between the states of the system that actually caused the significant change in behaviour.

This is due to the butterfly effect: “a small change in one state of a dynamic non-linear deterministic system can lead to large differences at a future state underlying the principle of sensitivity to initial conditions of such systems [142], [77]. Lorenz uses the well-known

metaphor (particularly popular in popularised science) for this kind of behaviour is the butterfly flapping its wings in Texas and causing a hurricane in China [142]. The butterfly flapping its wings (which seems that a very minor change in one state) has apparently caused a series of events that led to a large difference in a later state (hurricane).

2.5.5 Lyapunov exponent in measuring sensitivity to initial conditions

In practice, the sensitivity to initial conditions is measured using the Lyapunov exponent [63]. We proceed by explaining how Lyapunov exponent functions mathematically.

We represent a trajectory of a dynamic system in phase space as x_n . This trajectory is composed of a point sequence in phase space as the following in Equation 2.7:

$$x_1, x_2, x_3, \dots, x_n \quad (2.7)$$

We now consider a second trajectory $x_{n'}$ which is also composed of a point sequence that consists of a slightly modified initial state from the previous trajectory. This trajectory is infinitesimally close with the previous trajectory. This point sequence is defined in Equation 2.8:

$$x_{1'}, x_{2'}, x_{3'}, \dots, x_{n'} \quad (2.8)$$

The distance in phase space between these two trajectories, or in other words the rate of separation between them, is defined as δ . It consists of a different value for each respective states of the two trajectories over time. The infinitesimally close initial state of separation (at $t=0$) can be denoted as δ_0 and at time t as δ_t [242]. A graphical representation of the two trajectories and the distance δ between them is illustrated in Figure 2.3. The two trajectories diverge at a rate ‘ t ’ given by the Equation [88]:

$$|\delta Z(t)| \approx e^{\lambda t} |\delta Z(0)| \quad (2.9)$$

This rate of separation between the two trajectories at rate “ t ” can be approximately calculated using the Lyapunov exponent (which is derived by Equation 2.9) [12] using the Equation 2.10.

$$\lambda_t = \frac{1}{t} \ln \frac{|\delta Z(t)|}{|\delta Z_0|} \quad (2.10)$$

The Lyapunov exponent values can range from $-\infty$ to $+\infty$. There are three possible ranges of values that the Lyapunov value can take [88]. We proceed by listing these three ranges of values and describe what they mean.

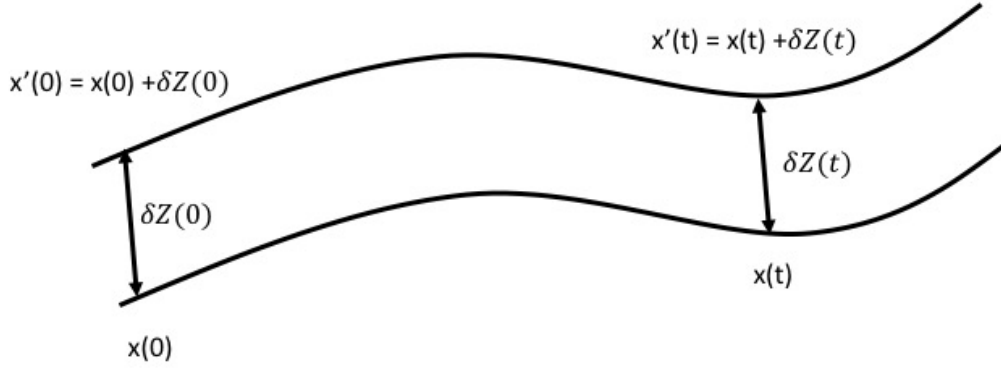


Fig. 2.3 Trajectories Separation in Phase Space by δZ

- Case 1: $\text{Lyapunov} < 0$: In cases which the Lyapunov exponent is negative, the dynamic system does not demonstrate a chaotic behaviour. This means that the separation δ between the two trajectories is not large enough to give a positive value for the Lyapunov exponent and hence, the two trajectories converge. As a result, the dynamic system demonstrates a more stable behaviour [88], [146], [50].
- Case 2: $\text{Lyapunov} = 0$: Rarely, the Lyapunov can be zero. This indicates that the system is in stable state and the states between the two trajectories are identical [238], [46], [146].
- Case 3: $\text{Lyapunov} > 0$: In cases which the Lyapunov exponent is positive, the dynamic system demonstrate a chaotic behaviour and its behaviour can no longer accurately be predicted. This means that the two trajectories exponentially diverge from each other, with the distance δ also being exponentially increased, thus producing a positive Lyapunov exponent [88], [146], [50].

2.5.6 Lyapunov Exponents in DDoS detection

As stated in [242], [129], [234] Lyapunov exponents can be estimated from motion equations of the dynamic system or from observed time series. In the scope of the thesis, we are

interested in the second approach. The network traffic state can be observed in the form of time series as the one in Equation 2.11:

$$A_1, A_2, A_3, A_4, A_i, \dots, A_n \quad (2.11)$$

To effectively detect any malicious deviations in traffic using Lyapunov exponents, the authors in relevant studies of work [50], [218] considered the two trajectories to be the actual traffic and some type of a prediction model of the traffic. The authors in [50], consider the following outcomes depending the value of the Lyapunov exponent:

- Case 1: $\text{Lyapunov} < 0$: In cases which the Lyapunov exponent is negative, the network traffic state does not demonstrate a chaotic behaviour. This means that the separation δ between the two trajectories, in other words the error value, is not large enough to give a positive value for the Lyapunov exponent and hence, the two trajectories converge. Thus, the network traffic demonstrates a stable and predictable behaviour. In practice, this means that the predictions are accurate enough, as the error between the actual traffic and its prediction is not exponentially increased and there is no indication of a DDoS attack present.
- Case 2: $\text{Lyapunov} = 0$: In such a case, the network traffic is in a stable state and there is no indication of a DDoS attack present.
- Case 3: $\text{Lyapunov} > 0$: In cases which the Lyapunov exponent is positive, the network traffic state demonstrates a chaotic behaviour and can no longer be predicted. This means that the separation δ between the two trajectories, in other words the error value, is increased exponentially and the two trajectories also exponentially diverge, thus giving a positive value for the Lyapunov exponent. In practice, this means that the predictions are no longer accurate, as the error between the actual traffic and its prediction is exponentially increased. This is a strong indication of a DDoS attack being present [88], [146], [50].

2.5.7 Machine Learning Techniques

The most popular and widely known definition for machine learning was given by Tom M. Mitchell as the following: " A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." [160].

Machine learning is a subset of artificial intelligence and is mainly linked with pattern recognition and computational learning theory. In practice machine learning deals with the

design and construction of algorithmic models trained through a series of data so they can make accurate predictions about future events or behaviour. Due to its popularity, machine learning is used a multiple and different areas of computer science. Some examples include email filtering, optical character recognition, user behaviour analytics, bioinformatics, information retrieval, text analytics, robotics, natural language processing, speech recognition recommendation systems and intrusion detection. Machine learning algorithms consist of two main phases: training and evaluation output. The former denotes the training approach followed to construct the model and the latter denotes the desired output during the constructed algorithm's testing. Proceeding, we briefly describe the categories according to the respective categorisations.

Division of Machine Learning approaches according to type of evaluation output

- **Classification:** Each instance to be evaluated is mapped to a class from a set of at least two classes chosen by the constructed model.
- **Regression:** Each instance to be evaluated is mapped to a numerical output calculated from the constructed model.
- **Clustering:** Incoming instances are to be categorised to a specific cluster from which the constructed algorithm developed. In contrast to classification and regression, the classes are unknown.

Division of Machine Learning approaches according to training approach

- **Supervised Learning:** During training, the training set provided to the algorithm consists of only labelled instances. Examples of labels include "spam" or "no-spam" in a email spamming detection training set. Based on the training provided, the algorithmic model constructs a set of mathematical/statistical relationships where inputs are associated to their respective labels. After the training phase is finished, the algorithm is ready to start predicting/classifying incoming unseen data. Classification and regression problems can be solved using supervised learning [33].
- **Unsupervised Learning:** During training, the training set provided to the algorithm is not labelled. Therefore, the chosen algorithm makes assumptions about the data by extracting underlying patterns and trends. After the clusters are constructed, the algorithm receives incoming data and groups them into one of the clusters. In unsupervised clustering this can form the evaluation output as well as using an unsupervised

algorithm for classification e.g. clustering to classify malicious traffic to a specific attack category [33].

- ***Semi-Supervised Learning:*** In semi-supervised machine learning, a part of the training set is labelled. Hence, the algorithm makes predictions/classifications data based on a series of assumptions constructed.
- ***Reinforcement Learning:*** In reinforcement machine learning, the training set is given as feedback to the model using the reward-loss concept on successful and unsuccessful predictions/classifications. The algorithmic model continues being improved based on a dynamic system.

2.5.8 Regression Analysis used in Machine Learning

In statistical modelling, regression analysis denotes the investigation of a mathematical relationship between a dependent variable and a single or set of independent variables. Based on its functionality and similarity to forecasting algorithms, regression analysis is primarily used for approximately predicting future events. Using it, the relationships between independent variables and the dependent variable can be understood. Moreover, we can understand which specific independent variables significantly affect the dependent variable and how.

Before regression, the least squares method was proposed by Legendre in 1805 [135] and then by Gauss in 1809 [39]. Then, they worked together to decide on the orbits of bodies about the sun based on astronomical observations. This resulted into the birth of regression analysis. Regression is a popular method in machine learning and was successfully used in multiple fields such as business, economics, finance, engineering and computer science. Beyond forecasting, regression can be used for effective decision-making, error avoidance and identifying underlying patterns, trends and relationships between data. Different types of input data can be used in regression such as discrete values, time-series, growth curves and images graphs. Under certain circumstances, regression can work with missing values. Over the years, numerous types of regression models were proposed such as linear, multiple and multivariate. For modelling regression models, we define the following axioms and assumptions.

Modelling of Regression Models

Axioms:

1. The unknown parameters (e.g scalar or vector), are symbolised as b .

2. The independent variables are symbolised as x .
3. The dependent variable is symbolised as y .
4. A regression model is symbolised as $y \approx f(x, b)$

Assumptions:

1. The input sample characterises up to satisfactory degree the population.
2. The error generated from the predictions is random and has a mean of zero.
3. The independent variables are calculated without any error.
4. There is a linear independence between the independent variables.
5. The errors generated from the predictions are uncorrelated with each other.
6. The generated error variable's covariance is constant across all observations.

Linear Regression

Linear regression denotes a linear degree relationship between a dependent variable y and one or more independent variables x [56]. A simple linear regression is the simplest form and consists of only one independent variable. If a linear regression model consists of more than one independent variables then it is called a multiple linear regression model. A linear regression model assumes that observations denote random deviations expected from a logical relationship between the independent variables and the dependent variable. A representation of it is in Equation 2.12, where y is the dependent variable, x is the independent variable, b_0 is the constant, b_1 is the coefficient and ε_i is the error for n data.

$$y_i = b_0 + b_1 x_i + \varepsilon_i, i = 1, \dots, n \quad (2.12)$$

A representation of multiple linear regression is in Equation 2.13, where y is the dependent variable, x_s are the independent variables, b_0 is the constant, b_1 to b_p are the coefficients and ε_i is the error for n data.

$$y_i = b_0 + b_1 \cdot x_{i1} + b_2 \cdot x_{i2} + \dots + b_p \cdot x_{ip} + \varepsilon_i, i = 1, \dots, n \quad (2.13)$$

Polynomial Regression

Polynomial regression, denotes a regression model where the relationship between the dependent and the independent variables is not linear, but of a higher degree (polynomial) [56]. A representation of it is in Equation 2.14, where y is the dependent variable, x s are the independent variables, b_0 is the constant, b_1 to b_p are the coefficients and ϵ_i is the error for n data.

$$y_i = b_0 + b_1 \cdot x_{i_1} + b_2 \cdot x_{i_2}^2 + \dots + b_p \cdot x_{i_p}^p + \epsilon_i, i = 1, \dots, n \quad (2.14)$$

Logistic Regression

Logistic regression was firstly proposed by David Cox in 1958 [55] and denotes a special type of regression model. In contrast to linear, multiple and polynomial regression models, where the dependent variable was continuous, in logistic regression the independent variables are continuous but the dependent variable is binary. Hence, it is suitable for problems which the dependent variable is dichotomous. Therefore, logistic regression can be effectively used in classification problems. The dependent variable can only take binary values, "0" or "1" which can be mapped to categorical labels such as pass/fail, disease/no disease, intrusion/no-intrusion and so on. Furthermore, logistic regression can be extended to consider more than two categorical values for the dependent variable. In such cases, the regression model is called multinomial logistic regression model. The predicted values are mapped to probabilities using the sigmoid function in Equation 2.15, where $p(x)$ is the final probability estimate between 0 and 1, x is the input to the function and e is the base of the natural log.

$$p(x) = \frac{1}{1 + e^{-(y)}} \cdot y = \beta_0 + \beta_1 \cdot x \quad (2.15)$$

Using the sigmoid function, the values can be transformed in the range of 0 to 1. A sigmoid function is used to transform a natural number to a probability. The function is called sigmoid because when graphically represented, it forms an "S" shape. In binary logistic regression, for incoming instances to be classified as either one class or the other, a threshold is selected. The instances are classified based on Equation 2.16.

$$p(x) = \begin{cases} > 0.5, & \text{class 1.} \\ \leq 0.5, & \text{class 0.} \end{cases} \quad (2.16)$$

Logistic regression has an approximating nature [32]. The model when presented with a vector of inputs, outputs an approximation for that specific input. Hence, there is always an

error between the approximating value predicted and the actual value for each input. The goal of training a logistic regression model is to find the most optimal coefficients: the set of coefficients that give the minimum error across all the instances of the training set provided. To achieve this, cross entropy function in Equation 2.17 in combination with the gradient descent function [32] in equation 2.18.

In detail, the cross entropy function measures the difference between the predicted and the actual value for each instance of the training set. The result is a number which represents the error generated with the current set of coefficients. The total cross entropy is calculated when all the errors were calculated for all instances of the training set using 2.17. Equation 2.17 denotes the sum of all errors between the actual values a_i and the predicted value p_i for every instance i from a total number of n instances in the dataset.

$$Cross = -\frac{1}{n} \sum_{i=1}^n a_i \cdot \ln(p_i) + (1 - a_i) \cdot \ln(1 - p_i) \quad (2.17)$$

During training, the goal is to optimise the coefficients so they can produce the smallest error possible. Hence, to minimise the cross entropy error we use the gradient descent function in Equation 2.18. The gradient descent function is an optimisation function that aims to find the minimum point of a function, in our case the cross entropy function. To achieve that, it makes usage of the partial derivatives for every coefficient. As a result, new coefficients are calculated at the end of every iteration that are more prone to give a more accurate value for the instances in the dataset. This procedure is repeated until the most optimal coefficients are found, in other words until the minimum of the cross entropy function value is found. In Equation 2.18 each new coefficient is calculated through subtracting the partial derivative of it times the learning rate γ from the partial derivative. The learning rate γ determines to what extent the newly acquired observations override the old ones.

$$weight_{i+1} = weight_i - \gamma \partial(weight_i) \quad (2.18)$$

2.6 Dataset Imbalance Prevention Techniques

Ideally, a training set used to construct an algorithmic model would be balanced with regards to the number of instances across the different categories present. When the difference in number of instances between classes is relatively small, the machine learning or statistical algorithm's performance is not significantly impacted. However, when this difference is large, the majority of algorithms are likely to classify an unknown instance as one belonging to the majority class.

There are several approaches to mitigate this issue which involve the pre-processing of the training set. The first method is undersampling, where random instances from the majority class are removed and all instances from the minority class remain intact. The second method is over-sampling, where new instances for the minority class are generated and added to the training set to even out its difference in size from the majority class whose instances remain unchanged. The third method is hybrid which is a combination of the two.

In our research, we chose the over-sampling method, since it is generally known that the majority of machine learning and statistical algorithms require a large training set to accurately construct their model. Specifically, we chose the synthetic minority oversampling technique (SMOTE) algorithm [1]. SMOTE algorithm works by creating convex combination of near in-distance existing instances in the training set instead of just copying existing instances. In more detail, SMOTE takes a sample of feature space of the minority class and its neighbours and combines them to generate new instances based on the minority class and its neighbours.

2.7 Summary

In this chapter we presented the background knowledge and mathematical concepts required for the thesis. We described how a smart home and the smart grid function and operate on their own as well as how they interact and communicate with each other by analysing possible architectures of them.

We proceeded with introducing the most notable vulnerabilities in IoT networks, such as the smart home-smart grid, analysing different types of vulnerabilities that could be exploited and lead to the compromise of the network by adversaries. We described and analysed an extended list of IoT attacks that can occur, focusing on DDoS attacks in multiple layers on the TCP/IP stack.

The next step was to outline how IDS systems work by firstly describing what kind of data they collect to analyse further, then what kind of detection techniques are used to classify a behaviour as malicious or not and finally how a decision is taken in an IDS. In addition, we included the most popular IDS architectures used. Finally, we described popular detection techniques used on IDS systems, focusing on forecasting techniques, chaos theory, regression approaches and machine learning techniques.

Chapter 3

Literature Review

3.1 Overview

As defined in Chapter 1, the hypothesis consists of the following question:

"Could a lightweight multi-layer IDS accurately detect application layer flooding and slow-rate DDoS attacks in the smart home-smart grid network and correctly differentiate it from legitimate traffic?"

Based on the hypothesis and objectives defined in Chapter 1 we aim to design and implement an IDS that has the following requirements:

1. The IDS should detect application layer flooding and slow-rate DDoS attacks with high true positive (TP) and low false positive (FP) rate.
2. The IDS should be smart and lightweight regarding its detection algorithms deployed as it is intended for the smart home-smart grid network which consists of mainly resource-constrained IoT devices.
3. The IDS should have a multi-layer detection where an additional layer is only used to improve detection rates and also provide multi-attack classification capabilities.

Based on the characteristics provided, we are motivated to conduct an extended literature research using the following approach. Initially, we research on IDS studies that are focused on detecting application layer DDoS attacks. Based on them, we will gain valuable information regarding the challenges application layer DDoS attacks pose, what techniques are particularly effective against them and what features are assisting in differentiating between them and normal traffic.

Proceeding, as stated in Chapter 2, machine learning is a popular set of techniques used in the detection of cyberattacks and intrusions, including DDoS attacks across all layers.

Hence, we conduct an in-depth investigation of how machine learning and other intelligent techniques, such as regression, forecasting and chaos theory, are applied in DDoS detection. Through these studies we aim to become familiar with the latest trends in the literature regarding DDoS detection using smart and sophisticated techniques. Our main focus should be smart homes, the smart grid and other resource-constrained networks but we also included studies that target traditional ICT networks as well.

Our proposed IDS system should be lightweight so it does not negatively impact the resource-constrained devices in the smart home-smart grid network. Therefore, we proceed by focusing on studies that use lightweight techniques to detect DDoS attacks that may or may not make usage of lighter machine learning models. Through the review of this set of studies, we aim to understand more about how lightweight techniques are used in DDoS detection and which are the most effective compared to heavier machine learning techniques. Finally, we review related IDS studies that follow a multi-layer approach. Through them we aim to understand in depth the following:

- How different layers interact with each other.
- Under what circumstances the different layers interact and communicate with each other.
- What type of information and data is exchanged between the different layers.

Through the acquired information we aim to understand how various techniques proposed in literature can assist in our problem statement. In summary, we group the relevant studies, starting from the least recent one, in the following way presented below.

- IDS studies conducted against application layer DDos attacks.
- IDS studies using machine learning and other A.I. techniques.
 - IDS studies focusing on smart homes and/or the smart grid.
 - IDS studies focusing on other resource-constrained networks(e.g. VANETs, WSNs).
 - IDS studies focusing on lightweight detection techniques.
- IDS studies that follow a multi-layer/multi-detection approach.

3.2 IDS systems studies on application layer DDos attacks

IDS systems detecting application layer DDoS attacks are relatively less studies compared to IDS systems detecting DDoS attacks in lower layers such as the TCP layer. The majority of research was focused on detecting DDoS attacks where human behaviour is directly involved.

This is evident in [240], where the authors implemented an IDS to detect malicious behaviour on user browsing behaviours, more specifically application layer DDoS attacks. To achieve that, they modelled the user's behaviour using semi-hidden Markov models. For modeling the overall behaviour user's behaviour characteristics were captured. Examples included number of clicks, links and hyperlinks clicked, series of URLs accessed, time spent on each page and so on. In an attempt to reduce the model complexity, a variation of the M-algorithm is used. To detect the malicious behaviour the user's HTTP request sequence fitting to the model is used. In their experiments, the authors reported the rates approximately being: TP rate with 99%, accuracy rate with 98% and FP rate with 1.5%.

Similarly, the authors in [250] used average-linkage clustering analysis to detect application layer user-based DDoS attacks. The features used were once again user-based characteristics such as IP timestamp, user's request sequences, how many objects are present on each session, reply size and website accessed popularity. The traffic collected from the authors, consisted a total of 47,748 requests from which: 46,014 GET, 1622 POST, 107 HEAD, and 6 invalid. Two datasets were constructed from the captured traffic. Clustering algorithm achieved a 93.16% TP rate for the first dataset and 90.59% for the second dataset.

The authors in [243] aimed to detect application layer DDoS attacks, specifically session and request flooding and asymmetric attacks, by monitoring user behaviour. They argued that due to lack of dataset available they created their own using LOIC and Goldeneye tools [92]. They used 17 features (based on user behaviour) to detect malicious behaviour. In their IDS, the authors used principal component analysis (PCA) for feature reduction and logistic regression for malicious activity detection. Six series of experiments were conducted: a training and testing set for each of the three attacks. Average results were the following : 1.41% FP rate and 98.64% TP rate.

The authors in [7] aimed to detect application layer user-based DDoS using the naive Bayesian multinomial algorithm. In contrast to the majority of other studies, they do not use user behaviour-based features, instead they use the number of flows and number of packets. The proposed solution achieved an approximate TP 94% and FP 4% rates.

The authors in [245] once again argued that it is difficult to detect user-based application layer DDoS attacks due to lack of data and features. Hence, it is hard to distinguish normal from malicious user behaviour. Therefore, the authors put effort in finding relevant and important features. The features proposed were: source IP address, timestamp, timezone,

URL, server response code, number of bytes sent, refer and user-agent. The authors used deep learning to identify these features, specifically stacked autoencoder deep learning architecture. Proceeding, they used logistic regression for classifying incoming traffic. They generated a dataset consisting of request and session flooding and asymmetric attacks. Asymmetric attacks constantly construct requests of high content consumption. Examples include database queries, images or large files request, password authentication and calculation. In practice, such attacks include: randomly generated request sequence attack, the cycle of pre-set sequence attacks, blocking and replay attacks, remote control of infected host attacks as the authors explained. For request flooding, the IDS achieved a 99.57% TP rate and 1.15% FP rate, for session flooding 98.87% TP rate and 1.26% FP rate and for asymmetric attack a 98.53% TP rate and 1.40% FP rate.

The authors in [171] aimed to detect application layer DDoS attacks, based on user behavior anomaly detection. They monitored the IP address, the URL and relevant timestamp. To detect the malicious behaviour, they have used the principal component analysis (PCA) as an anomaly detection method. The PCA algorithm was modeled through 2047 normal instances. During testing, there were 3917 attack and 1133 legitimate instances. In their experiments the authors reported a 99.64% overall accuracy.

PCA can also be used as a classification technique in addition to a dimensionality reduction algorithm. Specifically, it highlights relationships between variables (features) and cases (instances). For this to be achieved, the features and the instances are graphically represented in a multi-dimensional space. If the number of dimensions is larger than two, the projections of the points (features or instances) are reduced to a two-dimensional representation [213], [185].

The authors in [230] once again highlighted the challenge of detecting application layer DDoS attacks without a proper dataset available. Therefore, they created their own dataset to detect application layer DDoS flooding and asymmetric attacks. They divided these two types of attacks and used multi-features information entropy for flooding and second-order hidden Markov models for asymmetric attacks. Specifically, The concept of second order hidden Markov models was firstly introduced by [179]. Their reasoning was that first-order hidden Markov models contain the assumption that the state transition and output observation are dependent only on the previous state. This can cause issues on real applications [134]. The second order hidden Markov model relies on multiple previous states. More information regarding how it works can be found on [134].

The features used were requests count, session build-rate, session duration, request rate, page proportion and static Dyn rate. The authors constructed their own dataset using HOIC

[92] tool and the DePaul University website user access logs. They reported a TP rate 88.7% and a FP 8.9% rate.

Similarly, the authors in [34] aimed to detect application layer DDoS attacks based on user behaviour. In their study, they generated their own application layer DDoS dataset using tools like LOIC and GoldenEye [92]. They extracted the following features: mouse movement, clicks, highlights, drag and drop, wheel and scroll, key press and release events. When a request is received, it is analysed and the relevant features are extracted. If these features are not present, then a CAPTCHA is generated to be solved. If it is not solved, then the request is blocked. In their experiments, they report a 100% TP rate.

3.3 IDS Studies using machine learning and other A.I. techniques

Machine learning and other intelligent A.I. techniques, such as regression, forecasting, chaos theory are popular choices in DDoS detection. In this section we begin by reviewing machine learning and A.I. techniques notable studies for the smart home and/or the smart grid and then we expand our research for other low-resources networks such as WSNs, VANETs and IoT. Finally, we review a set of notable IDS studies that use lightweight machine learning and intelligent techniques for detection.

3.3.1 IDS Studies using machine learning and other A.I. techniques for the smart home and/or the smart grid

Attacks spreading from one network to the other in smart grid are considered a serious threat. Hence, the authors in [255] designed a new architecture for a distributed-hierarchical anomaly intrusion detection system. Their proposed system can be applied to the three-layer communication network (home area network (HAN), neighbourhood area network (NAN) and wide area network (WAN)) of the smart grid ecosystem. The authors' proposed system, consisted of a distributed IDS using support vector machines (SVM) and artificial immune systems (AIS) to detect KDD99 dataset attacks. The architecture proposed consisted of the three layers, HAN, NAN and WAN which is described above (smart home architecture). Due to the distributed nature of the proposed IDS, attacks on all layers were considered. In all three layers, HAN, NAN and WAN, probing and DoS attacks were considered. Additionally, in the WAN layer R2L (Remote-to-Local) and U2R (User- to-Root) attacks are considered all from the KDD99 dataset. The algorithms achieved satisfactory results in classifying the data, with SVM being the most accurate with only 0.67% FP rate and 97.85% TP rate. The

two AIS algorithms (CLONALG and AIRS2Parallel) used, achieved 0.7% and 1.3% FP rates and 78.98% and 73.7% TP rates respectively.

The authors in [26] effectively stated that one of the most critical nodes in the smart grid is the data concentrator unit. The data concentrator unit is responsible for collecting data from an assigned neighbourhood's smart meters and forwarding them to the AMI headend. Hence, they proposed a distributed IDS installed at every data concentrator unit to protect them from wormhole attacks. For detecting this attack, they made usage of the hop count a packet has to make. They used Euclidean distance to estimate the hop count number expected and whatever exceeds these limits is classified as malicious. To evaluate their IDS, the authors simulated three types of geographical areas consisting of smart meters and the data concentrator unit: suburban, rural and urban areas. In their scenarios they used variations of wormhole attack, including the delta wormhole attack and varied number of attackers. They reported an overall result of their scenarios for each area. For the suburb area, they reported the FP and TP to be 4.4% and 94% respectively. For the urban they reported 4.8% and 96.5%. Finally, for the rural area they reported are 0% and 94.2%.

The authors in [13] proposed a distributed IDS to detect DDoS attacks, probe attacks and combination attacks of these two in AMI systems. To detect these attacks in this heterogeneous and distributed network and manage the malicious behaviour more effectively they install an IDS on the headend and on each data concentrator. The authors generated a dataset containing normal and malicious traffic based on the KDD 99 dataset. The training set consisted of 5500 TCP packets and the testing set consisted of 2750 packets. In both training and testing 75% was normal and 25% was attack. A total of 27 features were used that have to do with TCP such as flags, window, checksum along with IP source and destination. In an attempt to reduce complexity, the authors used PCA to reduce the number of features from 23 to 10. For detection, they used a variation of the K-means clustering, called the mini-batch K-means clustering. The authors conducted 3 series of experiments, one for DoS attacks, one for probe and one for their combination. For the DoS attacks, the TP was 100% and the FP initiated from 80% but was minimised through time to nearly 0%. For the probing attacks the mini-batch algorithm is inconsistent regarding the TP and FP rates, as it ranges from 100% to nearly 0%. In the final combinational attacks scenario, the mini-batch algorithm achieves 100% TP and 0%. They compare the mini-batch to the regular K-means to show the former achieved more accurate results.

The authors in [122] constructed a lightweight IDS to detect DoS, portscan, SSH brute-force and HTTPerf attacks against SCADA systems and other industrial control systems. To achieve, that they decided to monitor the current electricity flow on the system and compare it using a similarity measure called sum function. The IDS was trained for 72 hours. During

evaluation, each attack lasted for 14 hours containing 11 different attacks per hour. The IDS's TP rate was reported 100% and FP rate was 0.13%

The authors in [18], aimed to detect scanning and DDoS (UDP and TCP flooding) attacks in smart home IoT networks. They constructed an IoT network (consisting of routers, printers, amazon echo dots, smart plugs, cameras, apple TV, Iphone 6, door/window sensor and motion sensor; all correspond to a Sky Router). They deployed a single attacker in the network, conducting the various attacks. The authors generated a dataset which consisted of 4 days: 2 with completely legitimate traffic and 2 that consisted of attacks. The features used were attack start time, destination IP, protocols used and size of the packets transmitted. The authors used a number of different machine learning algorithms using 10-fold cross validation, with naive Bayes achieving the best results. In detail, they achieved a TP rate of 97.7% for quick scans, 95.5% for regular/intense scans, 68.8% for SYN flood and 68.8% for UDP flood.

The authors in [64] supported that in devices deployed smart home environments have specific communication procedures. This potentially results in predictable consumption patterns. Based on these assumptions, they proposed an IDS that used machine learning to monitor the power consumption of smart home devices to detect TCP SYN flooding DDoS and brute-force attacks. They used a Raspberry Pi 3 model B+ and a camera attached to it. For their evaluation, the machine algorithms used were single class SVM, binary class SVM (both used radial basis function (RBF) kernel), K-nearest neighbors, random forest and logistic regression. SVM (both single and binary) performed best with a 94% TP rate, with random forest performing the least best with 92.5% TP rate.

3.3.2 IDS Studies using machine learning and other A.I. techniques for other resource-constrained networks

Beyond the smart home and/or the smart grid network, other notable IDS studies were conducted in other resource-constrained networks. Examples included as IoT networks, vehicular area networks (VANET), Ad-Hoc, mobile area networks (MANET) and wireless sensor networks (WSN). In this subsection, we provide a set of notable IDS studies focusing on such networks.

The authors in [104] effectively supported that IoT devices are easy to be compromised due to their weak security measures. As a result, they become part of botnets used for DDoS attacks. Hence, they proposed a new innovative IDS, called D-Pack, that uses a convolutional neural network (CNN) and autoencoder unsupervised deep learning module for auto-profiling traffic patterns and filtering DDoS attacks. To evaluate their IDS they used

the USTC-TFC2016 and Mirai-RGU datasets. In their results they reported a TP rate ranging from 99.39% to 100 and a FP rate of 0.83%.

Highlighting the need for effective security in IoT networks the authors in [152] also proposed a deep-learning method, specifically a deep belief network (DBN) algorithm model to detect malicious behaviour from the CICIDS 2017 dataset. They reported their proposed system achieving a 99.37% accuracy for normal class, 97.93% for botnet class, 97.71% for brute force class, 96.67% for DoS/DDoS class, 96.37% for infiltration class, 97.71% for ports scan class and 98.37% for web attack.

In [115] the authors made usage of logistic regression to detect DoS and black hole attacks along with messages eavesdropping in VANETs. They used features directly related to VANETs including packets delivery ration between vehicles, number of attacked vehicles and numbers of attackers per vehicle. They generated traffic using OMNET++ and Sumo0.15 to construct the logistic regression model. They reported a TP rate of 82% for their proposed IDS.

The authors in [99] used the multi-layer perceptron (MLP) architecture of neural networks to detect DDoS attacks in IoT networks. The authors created a dataset to train and evaluate the MLP neural network. This environment had 4 IoT nodes and a fifth one acting as a server for them. The DoS attack was performed by one additional malicious node, which sent over 10 million packets. The DDoS attacks were performed by 3 nodes sending over 10 million packets. Those packets used the UDP protocol. They trained the MLP neural network with 2313 samples, validated with 496 samples and tested with another 496 samples. Of the total dataset, 2121 were malicious and 1184 were legitimate instances. In their results they reported a TP 99.4% rate and a 0% FP rate.

The authors in [116] proposed a lightweight IDS that can be installed in IoT devices to detect DDoS attacks. They monitored host-based features from NTP servers. Network time protocol (NTP) is used for clock synchronization between computer systems over packet-switched, variable-latency data networks. The features considered were delay time, response delay time and offset clock. To detect DDoS attacks they calculated the mean and standard deviation and used the central limit theorem to classify the behaviour as malicious. The attacks duration was approximately 20 minutes. In their results, the TP rate was reported as 100% and the FP rate was 16.66%.

The authors in [203] used machine learning approaches to detect wormhole DDoS Attacks in IoT networks. They presented three centralised approaches, one which consists of decision trees, one consisting K-means clustering and one that uses both approaches. Decision tree algorithm is trained through learning the safe distance between two neighbourhood routers. The decision tree is constructed to decide if two routers can become neighbours or are

attacked. K-means is used to identify the safe zones, in an attempt to reduce false positives. The hybrid consists of both algorithms used together. In their experiments, they used various IoT topologies, including mesh, ring and star. For each topology they used various number of nodes including 10, 50, 100 and 200. The TP rate for K-means was 70-93%, for decision tree was 71-80% and for the hybrid approach is 71-75%. Even though the hybrid approach achieves less detection rate than the individual approaches, they have much higher FP rate than the hybrid approach.

The authors in [85] aimed to detect flooding, wormhole and blackhole attacks using machine learning techniques in IoT networks. They used NS-2 to generate a relevant dataset and to conducted the experiments using Weka toolbox [144]. They used machine learning techniques such as naive Bayes, decision trees, random trees and Bayesian networks. For blackhole attacks, naive Bayes achieved a TP rate of 75.79%, decision tree 98.92%, random tree 74.76% and Bayesian networks 77.16%. For the flooding experiments the following TP rates were achieved: naive Bayes 77.38%, decision tree 72.63%, random tree 72.63% and Bayesian networks 81.05%.

The authors in [256] constructed an IDS to detect cyber attacks in IoT networks. For their experiments, they used KDD99 dataset. They used PCA to reduce the number of features and K-nearest neighbour (KNN) and softmax regression as the detection algorithms, separately. The dataset consisted of 490,000 from which 60% was used for training and the remaining 40% for testing. In their feature reduction experiments, the authors achieved three different types of results: one with 3 features, one with 6 and one with 10 features. However, for the evaluation of the KNN classifier they used 3 and 6 features. For the 3 features, KNN achieved an accuracy rate of 85.24% and for the 6 features 85.19%. The softmax regression achieved an 84.50% and 84.40% respectively. Softmax regression has a $O(m * n)$ complexity, where n is the number of features and m is the amount of information which comes with former iterations (between 5 and 20). Therefore, the complexity is $O(1)$. For KNN, the training phase has $O(n * \log n)$ complexity. For the testing phase it is $O(n + n * \log n)$. It is significantly higher than softmax regression. The FP rate for softmax was 5%, for both 3 and 6 features. No information about the FP rate of KNN was given. During testing, 144,021 instances were classified. Softmax regression required 5.72 seconds while KNN required 507.31 seconds.

Since IoT devices are high in number and heterogeneity, software defined networking (SDN) would be a notable solution for their effective management and monitoring. Hence, the authors in [30] proposed an IoT IDS called SoftThings for SDN-based IoT environments to detect DDoS attacks, specifically TCP and ICMP flooding. In more detail, a cluster-based IDS is proposed, where a cluster-SDN controller is installed in every cluster and a master

SDN controller is at the top of the IDS hierarchy. At regular intervals, the cluster SDN controller update the master SDN controller for any potential changes observed. Then the master SDN controller classifies the incoming traffic as malicious or normal. The detection algorithm used both linear and non-linear (RBF Kernel) SVM. The features used were: number of sent requests, number of failed authentication attempts, source of requests and bandwidth consumption device usage at different periods. The authors conducted three scenarios to evaluate their IDS accuracy. The first scenario consisted of TCP flooding, with the linear SVM achieving 94% precision and 92% TP rate. The non-linear SVM achieved 98% precision and 97% TP rate. The second scenario consisted of ICMP flooding, consisted of linear SVM achieving 92% precision and 88% TP rate. The non-linear SVM achieved 97% precision and 96% TP rate. The third scenario consisted of a DDoS attack, with the linear SVM achieving 94% precision and 93% TP rate. The non-linear SVM achieved 98% precision and 95% TP rate.

The authors in [11] supported the need for lightweight IDS in WSNs to detect DDoS types of attacks. In detail, the authors focused on DDoS attacks in lower layers to detect constant, random and reactive jamming and blackhole attacks. To achieve high detection rate without creating additional overhead, they used logistic regression for detection, deployed on each node of the WSN. To evaluate their proposed IDS, they conducted a series of experiments, in which each attack occurred for 15 minutes and it is repeated 5 times during each scenario. Regarding detection, in the constant jamming attacks the TP rate is approximately 95%, for blackhole attacks approximately 95% and for blackhole and random jamming approximately 60%. Regarding FP rate, in the constant jamming attacks the rate is approximately 0.025%, for blackhole attacks approximately 0.25% and for blackhole and random jamming approximately 0.25%.

The authors in [107] proposed logistic regression as a lightweight and effective way installed in sensors in WSN networks to detect greyhole and blackhole attacks. The features used were number of packets dropped, data packets received and number of packets forwarded. During the evaluation of greyhole attacks the TP rate was 75%. No FP rate was achieved. For the blackhole attacks the logistic regression model managed to get a perfect detection rate with no FP rates. The authors also conducted experiments by conducting multiple attacks simultaneously. They achieve a TP rate of 91%.

The authors in [194] also used deep learning models to detect DDoS attacks from the CICIDS 2017 dataset for IoT networks, highlighting the importance of features used. Specifically they implemented a combination of a Jumping Gene adapted NSGA-II multi-objective optimization method for data dimension reduction and the convolutional neural network (CNN) integrating long short-term memory (LSTM) deep learning techniques for

classifying the DDoS attacks. Amongst other results presented, they reported a 99.03% TP rate.

The authors in [191] also used a deep learning technique to detect DDoS attacks in IoT networks. However they differentiated their approach, as they chose a semi-supervised technique, specifically the deep-layered extreme learning machine (ELM) neural network. To evaluate their proposed solution they made usage of the ISCXIDS 2012 dataset. In their results they report a 97.6% TP rate and a 1.8% FP rate.

3.3.3 IDS Studies using A.I. lightweight techniques

As already stated, there is an extended set of A.I. techniques beyond machine learning. Two notable examples are forecasting techniques and chaos theory, specifically Lyapunov exponents. Forecasting techniques and/or Lyapunov exponents were used frequently on their own or in combination with other techniques to detect DDoS attacks in various layers.

In this subsection we present the most relevant studies to our research that make usage of forecasting/Lyapunov exponents to detect DDoS attacks. We begin by reviewing IDS studies that solely use forecasting methods, and proceed with reviewing notable IDS studies that use a combination of forecasting and/or Lyapunov exponents with Lyapunov networks.

To investigate which forecasting method is best suitable for detecting DDoS attacks, the authors in [71] used a number of forecasting algorithms, including moving average, weighted moving average, simple exponential smoothing forecasting and linear regression to predict the intensity and size of TCP-based DDoS attacks such as SYN flooding, DNS flooding and ICMP flooding. They used a window of 60 seconds between predictions to monitor the number of packets. The two thirds of the attack, from a total of 53 minutes, was used for training and the remaining one third was used for testing. To evaluate which forecasting algorithm produces the lowest error, they made usage of the absolute prediction error. For the TCP SYN flooding 563 packets were generated at each second, for DNS flooding 8 packets and for ICMP 1 packet per second. Simple exponential smoothing forecasting was the best in detecting the intensity of the attack and size of DNS TCP and ICMP flooding. Linear regression was the most successful in detecting the size of the TCP SYN flooding. In overall, the algorithm achieved an error rate of 1%.

The authors in [169] used simple exponential smoothing forecasting with Chi-square and average of long-term activities to detect event-based attacks from the DARPA dataset. Also, using the training set they constructed a Markov chain model to predict normal activities. After a series of experiments, they concluded that the optimal value for a is 0.5. Through the experiments, they proved that simple exponential smoothing forecasting combined with the Chi-square achieved higher results with over 80% and a FP rate of less than 5%.

In [176], the authors proposed a linear exponential smoothing forecasting model to detect flooding and scanning DDoS attacks using bytes, flows and packets per minute. They reported a 24-hour training time and a window size of 5 minutes to accurately detect deviations in traffic. The α value was set to 0.05 and the deviation was measured using entropy. Through their experiments, their algorithm could detect the attack, which lasted for 11 minutes, 13 hours after the algorithm has initiated training. However, it misclassified instances both before and after the actual attack. In total, 11 predictions misclassified normal behaviour as attack out of 288. This gave a 3.82% FP rate.

The authors in [218] aimed to detect DDoS attacks using forecasting and Lyapunov exponents. They measured the packets generated from each IP address. They used an autoregressive integrated moving average (ARIMA) forecasting model to predict traffic after using a training time of 600 minutes. At every time interval (300 seconds), an error was generated from subtracting the inverse box transformation of the relevant forecast from the actual value of the packets number. The error value was used as an input to the Lyapunov exponent formula. If it was positive, a set of rules constructed classified the traffic as D/DoS or legitimate. Their detection algorithm achieved a 94.4% TP rate and a 0.1% FP rate.

In [148], the authors made usage of the simple exponential smoothing forecasting algorithm to detect variations of TCP SYN flooding attacks. These attacks can be of low or high intensity. After a series of experiments, they reported the optimal window size to be 5 minutes and the α value to be 0.98. To conclude on whether there was an attack or just a normal temporary anomaly in the system they weight the forecast and compare it with the actual value. The optimal weight to give to the forecast is 0.5. If more than a certain number of weighted forecasts are less than each of their respective actual values then an alarm is raised. The optimal value for the number of consecutive anomalies is 4. In their results, the authors reported a TP rate between 60-85% and a FP rate of 30-60% for low intensity attacks. For high intensity attacks, they reported a 100% TP rate and a FP rate of 30-60%.

The authors in [199] combined simple exponential smoothing forecasting and Shannon's entropy for outlier detection in IoT networks. To detect the outliers, they monitored a set of different traffic characteristics including destination/source IP address distribution, flow asymmetry, inter-packet arrival, packet size and volume. They reported an overall accuracy ranging from 93-95%.

As stated, forecasting methods and/or Lyapunov exponents were also used in combination with other techniques, particularly artificial neural networks to produce more accurate results. Some notable studies that follow this approach are briefly described below.

The authors in [51] used Lyapunov exponents but instead of any forecasting methods they used self-similarity to calculate DDoS attacks based on time-series. Proceeding, they

used these two techniques to train their ANN as they wrote in [47] and [239]. They reported the TP rate ranging from 88-94% and a FP rate ranging from 0.05-0.45%.

The authors in [47] used linear AR model to predict incoming traffic state and using Lyapunov exponents to classify it as malicious or legitimate. They evaluated their approach by using the KDD99 dataset which also includes DDoS attacks. Based on this approach they used the results to train an ANN in order to potentially increase its accuracy. The authors used DARPA 1998 and 1999 for training and evaluation of their detection algorithm. They reported their approach initially achieving a 71.84% TP rate and then at the second improved stage achieving a 93.75% TP rate.

In their second piece of work [239], they used another technique called coupled-cluster to combine it with Lyapunov exponents in assessing the error by the simple exponential smoothing forecasting method. However, instead of cumulative average they used simple exponential smoothing forecasting method. All these techniques were again combined with neural networks. They also increased the training size to 16000 samples. In their results they reported a TP rate of 98.04%, and a FP rate of 0.19%. They also reported that before using the neural networks their TP rate was 76.12%. In summary, they used simple exponential smoothing forecasting to make a prediction for the incoming traffic. The error was analysed in further, by calculating the Lyapunov exponent with the help of the coupled-cluster technique to study the average exponential rate of the network behaviour. The calculated Lyapunov exponent was finally classified by the back-propagation neural network.

In [147] the authors used Lyapunov exponents in combination with Tsallis entropy and autoregressive (AR) linear forecasting to detect DDoS attacks. They collected network traffic to extract the source and destination IP addresses and calculate their entropies. Then they used an AR model to generate dynamic equations. Next, they used the Lyapunov exponent to analyse the rate of exponent separation between source and destination IPs. The optimal detection time interval was 60 seconds. In their results, they reported the best rates to be 98.56% TP rate and 0.42% FP rate.

3.4 Multi-layer/multi-detection IDS studies

In this section, we review the most notable IDS studies that follow a multi-layer approach. There are multiple reasons a multi-layer approach is adopted in an IDS, with the most popular being to improve detection results. Some examples also include multi-attack classification, feature reduction and accuracy improvement through training. Hence, we group IDS studies based on the examples provided above and present them, starting from the earliest in date.

Multi-Layer Studies for improving detection and multi-attack classification

The authors in [187] designed a hybrid hierarchical IDS to detect cyberattacks from the KDD99 dataset. The IDS consists of three hierarchical modules: the attack classifier, the behaviour partition module and the final classifier. An enhanced version of the decision tree (C4.5 tree construction algorithm is used) is used for misuse detection and K-means clustering is used for anomaly detection. In the attack classifier, the enhanced decision tree was used to classify DoS and probe attacks. Whatever incoming traffic did not belong to either of these attacks, is processed further by the clustering algorithm. If it is normal, then it is classified as normal. If it is not normal, it is processed in further by another enhanced decision tree to classify it as either U2R or R2L. For training, 41 features were used. The hybrid classifier achieves an overall detection rate for DoS 83.59%, for probe 78.60%, for U2R 50.60% and for R2L 28.53%. Their system achieved a TP 97.3% rate and a FP 9.1% rate.

The authors in [54] proposed a hybrid, lightweight, distributed IDS to detect sinkhole and sleep-deprivation attacks in WSNs. The IDS includes a "central agent" that receives information from "local agents". The local agent encompasses a lightweight anomaly-based detection system on the nodes. The central agent encompasses a trained decision tree.

Each local agent consists of three components: The local packet monitor, the data collector, and the detection engine. The local packet monitor collects routing information from the network and sends detection-related data to the data collector. This component aggregates the monitoring data and forwards it to the detection engine. The detection engine works with threshold-based measures. If the incoming traffic surpasses these thresholds, the relevant information are sent to the central agent.

In their experiments, the authors tested their decision tree with different pruning approaches. Pruning aims to decrease the size of the trees. To achieve that, it removes sections of the tree that are not so important in classifying instances. In overall, this technique aims to reduce the complexity of the decision tree, and also improve the accuracy by reducing overfitting. The larger the pruning value is, the smaller and more concise the tree is. With 25 pruning, the accuracy is 99.546% and the FP rate is 0.0130%. With 50 pruning the accuracy and FP rates remain unaffected. Therefore, they concluded that increasing the pruning severity does not improve results.

The authors in [231] combined ant colony optimisation (ACO) and SVM to detect attacks from the KDD99 dataset. ACO was used a clustering technique. Both models were trained with 1239 instances, containing both malicious and legitimate traffic. During testing phase, various combinations of testing sets we reconstructed to evaluate the IDS's accuracy and resistance to bias. Both models are responsible for analysing incoming traffic. If both of

them classified it as normal, then the final classification is normal. If both of them classified it as malicious, the traffic is classified by ACO clustering as to what type of attack it is. If one of them classified it as an attack and the other as normal, it was considered ambiguous. The average TP rate across all 4 testing sets was 73.97% in contrast to each individual method used separately which achieved lower detection rate results.

The authors in [19] performed multi-attack detection using the UNSW-NB15 dataset. In detail, they constructed a multi-layer IDS. The first layer receives raw data to produce intermediate results and the second layer produces the final results by receiving the intermediate results from the lower layer and classified them. In the first layer, K-means clustering is used in the dataset to construct the centroids and group similar instances depending on their distance from their respective centroid. Then the Euclidean distances from the centroid and the nearest neighbour were calculated for every instance. Hence, a D-features dataset was reduced to one feature dataset so complexity, execution and run times are reduced and accuracy is improved. Finally, ensemble classifier was used for detection by receiving the new form of the dataset. In their results, the authors reported a 90% TP rate.

Multi-Layer Studies for Feature Reduction

To reduce complexity, various multi-layer studies were proposed which consisted the first layer that performed feature reduction in the first-layer. In that way, the detection algorithm is faster, less complex and potentially more accurate. In this section, we present the most similar studies to our research.

The authors in [110] used a variation of SVM called support vector relevance machine to detect attacks in the KDD99 dataset. To reduce the complexity, the authors made usage of the PCA algorithm for feature reduction. In the results section they compared the SVR with SVM using the same number of reduced features. The SVM achieved a 85.2% overall accuracy in 11.2 seconds and the SVR achieved a 90% in 0.98 seconds.

Similarly, the authors in [186] aimed to tackle overfitting, feature redundancy and high-dimensional features. Their difference though to other studies is that they used a single method for both feature reduction and classification. This decreases the overall IDS complexity and introduces a universal behaviour for multiple procedures. The method adopted was sparse logistic regression. This technique selects the discriminating features to model the approximation function and classify the data accurately. The dataset adopted for the evaluation of the IDS was KDD99 dataset. For training, they used 1,774,985 (of which normal : 812,814 and DoS: 947,267) and for testing 67,888. From a total of 41 features the sparse logistic regression managed to reduce them to 7. Based on these 7 features, sparse logistic regression achieved a TP rate 97.65% and a FP rate of just 0.34%. The training time

lasted for 11.6 seconds. The authors compared their proposal both with other related studies and other machine learning algorithms.

Multi-Layer Studies for Multi-Attack Classification

Towards improving the reduction but also achieving multi-attack detection various studies were proposed. In this section, we present a set of notable IDS studies starting from the study with the oldest publication date.

The authors in [20] designed a multi-layer IDS for multi-class attack detection in the NSL-KDD dataset. In detail, in the first layer they used PCA for feature reduction. The first layer (which contains the PCA algorithm) is only used during training. The PCA algorithm receives the training set and chooses the most important features. Therefore, it is not used during the evaluation. In the second layer, genetic algorithm was used to construct anomaly detection algorithms. These can differentiate between normal and abnormal behaviour present in the network. Genetic algorithm uses distance metrics to differentiate normal and abnormal. The authors used two distance metrics : Euclidean and Minkowski. The genetic algorithm is used to decide on whether incoming traffic is legitimate or malicious (not investigating the type of the attack). Genetic algorithm is used to generate the best fitted solutions (legitimate traffic instances) so they can be compared with incoming traffic. If incoming traffic does not match with any of the generated instances of the genetic algorithm, then it is assumed that it is malicious and it is forwarded to the different machine learning algorithms to be classified as a specific attack type.

In the final layer, the authors used several machine learning algorithms that receive the incoming malicious traffic to effectively classify it in the correct attack family. The machine learning algorithms used were naive Bayesian, MLP neural networks, decision trees, naive Bayesian tree, best-first tree and random forest tree. The authors conducted 4 types of experiments in which they had two independent variables: the number of train data and the distance metric used in the genetic algorithm.

In general, the tree-based techniques achieved the best results with the attacks that are represented most. In detail, decision tree achieved the best detection with DoS Attacks regardless of train size and distance metric, ranging from 81.72-81.97%. For probe attacks decision once again achieves the best results with 61.04-65.42% regardless of train size and distance metric.

For the attacks that are least represented in the dataset (U2R and R2L) the overall detection results were low for all the classifiers. Naive Bayes achieves the best detection results. In detail, for R2L it achieved a TP Rate 18-32.90% regardless of distance metric and train size. For the U2R attacks, it achieved a TP rate of 11.33-20.35% regardless of train

size and distance metric. Furthermore, naive Bayes TP rates for the DoS attacks were close to decision tree with 76.79-80.16% TP rate. For the probe attacks it achieved a TP rate of 53.31-55.14%.

The authors in [216] used two statistical methods, linear discriminant analysis and logistic regression to detect attacks from the NSL-KDD dataset. To reduce the complexity of the models, they made usage of the Chi-square to decide which features were the most important. The final set of optimal features was 23. They trained both approaches with 17389 normal instances, 70631 DDoS instances, 6372 probe instances, 1782 U2R instances and 2745 R2L instances. The test set consisted of 13,703 normal, 55,768 DoS, 2940 probe attacks, 4302 U2R and 3176 R2L instances.

In their results section, the authors compared their statistical algorithms with other popular machine learning techniques such as SVM, naive Bayes and decision tree. They conducted two series of experiments. In the first series, they performed multiple-attacks classification and in the second series, they performed binary classification. Regarding the multiple attacks classification, logistic regression achieved an average 98.74% TP rate and a 95.31% accuracy rate. Linear discriminant analysis gets an average 95.05% TP rate and a 95.05% accuracy rate. SVM achieved an average 97.56% TP rate and a 97.93% overall accuracy. Naive Bayes achieved an 87.97% TP rate and a 91.52% overall accuracy. Decision tree achieved an average 98.21% TP rate and a 98.29% overall accuracy. Based on the results, logistic regression was the most accurate classifier, generating the highest TP rate. Regarding the binary classification, logistic regression achieved a TP rate of 91.59% and accuracy rate 98.27%. Linear discriminant analysis achieved a TP rate of 90.89% and a 99.83 overall accuracy rate. SVM achieved a TP rate of 99.16% and accuracy tree 99.63%. Naive Bayes achieved a TP rate of 89.97% and 98.16% accuracy rate. Decision tree achieved a TP rate of 90.95% and a 99.85% accuracy rate. We summarise the results of the relevant study in the Table 3.1.

The authors in [177] constructed a dual two-tier IDS, the first two-tier for feature reduction and the second-tier for classification of the incoming traffic to detect attacks from the NSL-KDD dataset in IoT networks. For the feature reduction, they used PCA and linear discriminate analysis and for classification naive Bayes and KNN. The authors used two training sets: the first with 25,192 and the second with 125,973 instances. The testing set consisted 22,544 instances. The feature reduction algorithms decreased the number of the features from 41 to 35. The computational complexity of naive Bayes is $O(e * f)$ where e is the count of in dataset and f is the number of features and for KNN is $O(\log n)$. Regarding the results the IDS had a TP of 84.86% and 4.86% FP rate. Regarding multi-class detection

Table 3.1 Related Studies

<i>Algorithm</i>	<i>Binary Classification</i>	<i>Multi-Class Classification</i>
Logistic Regression	TP: 98.74% AC: 95.31%	TP: 91.59% AC: 98.27%
Linear Discriminant Analysis	TP: 95.05% AC: 95.05%	TP: 90.89% AC: 99.83
SVM	TP: 97.56% AC: 97.93%	TP 99.16% AC: 99.63%
Naive Bayesian	TP: 87.97% AC: 91.52%	TP: 89.97% AC: 98.16%
Decision Tree	TP: 98.21% AC: 98.29%	TP: 90.95% AC: 99.85%

rate the IDS achieved a correct detection rate for normal 94.43%, for probe 87.32%, for DoS 88.20%, for U2R 70.15% and for R2L 42%.

The authors in [221] used logistic regression for multi-class classification of cyberattacks from the NSL-KDD dataset. They used correlation-based feature selection to reduce the features used. They used 20% of the KDDTrain+ subset of NSL-KDD dataset to train and construct the model. To test the model they used two subsets of the NSL-KDD dataset, the KDDTest+ and KDDTest-21. For the KDDTest+ the FP rate for normal is 34%, for probe 16.8%, for DoS 12%. The overall FN rate is 21.4%. For the KDDTest-21, the FP rate was 74%, for probe 17.5%, for DoS 38.6%. The overall 46.9% was 46.9%. The authors compared their logistic regression model with decision tree and naive Bayes algorithms. For the KDDTest+ decision tree achieved a 65%, naive Bayes 74% and logistic regression 74%. For the KDDTest21 decision tree achieved a 38%, naive Bayes 50% and logistic regression 40%.

Multi-Layer IDS Studies to improve detection through training

Instead of improving the accuracy of their detection algorithm by directly improving the detection process, other studies improved their training process. In this section a set of notable IDS studies that improve detection rates through training in chronological order.

To improve the detection accuracy, the authors in [49] used the fast adaptive clustering algorithm so their hidden Markov model could detect KDD99 dataset attacks more effectively. Specifically, the fast adaptive clustering algorithm is used in the training phase to construct the clusters. Based on these clusters, the hidden Markov model is designed. The IDS was

trained with 39,012 normal instances and 9916 attack instances. During the testing phase, the clustering algorithm acts as a misuse detection technique and the hidden Markov model as the anomaly detection technique. In their results, they reported a FP rate of 1%. The misuse achieved a TP rate of 95% the anomaly 92% TP rate.

In [68] study the authors used decision tree and MLP neural network to detect attacks from the KDD99 dataset. In the training phase, they used 33,000 attack instances and 10,000 normal instances. Both of the models were trained with this training set and using all 41 features enlisted for the KDD99 dataset. At the end of the training phase, both of the machine learning models constructed a label for every instance. These labels were added as features in the dataset and a new version of it is constructed. The two algorithmic models were re-trained using this new training set. In the results section, the authors reported MLP outperformed decision tree. In detail, it achieved a TP rate of 99.3% and accuracy rate of 99.75%. Also, the results based on the enhanced new dataset were better than the original dataset.

In [155], the authors proposed two multi-layer architectures for protecting WSNs against various types of cyberattacks. Specifically, they used the KDD99 dataset for evaluating their proposals.

In the first architecture, the system consists of two layers. The first layer is deployed in the sensor. At this layer, the relevant data are collected, classified by a supervised algorithm and the result is sent to the upper layer. The upper layer, which is located in the sink node, consists of an unsupervised algorithm. The unsupervised algorithm receives the data from the lower level and classifies the incoming instance.

In the second architecture, the same procedure is followed. An additional layer is located at the top and consists of an additional layer which is located at the base station. It collects data from lower layers and further process them through another unsupervised algorithm. In both of these architectures, each algorithm at each layer constructs a set of detection rules at the end of the training phase. From this set of rules, the most relevant features are chosen (using entropy) along with their respective rules and construct an enhanced decision tree.

This enhanced decision tree uses the J48 algorithm. The enhanced J48 decision tree algorithm combines the knowledge acquired from the multi-layer architecture algorithms and in the testing phase makes the final decision. In both architectures, the supervised algorithm used is the Gini index algorithm and the unsupervised algorithm clustering. In their results section, the authors reported a sensor level detection accuracy 95.08%, a sink node detection accuracy 84.15% and the enhanced J48 decision tree 98.89%. The training phase consisted of 10% of the total size of the KDD99 dataset. For the supervised algorithm 14 features were selected and for the unsupervised 15 features were selected. The final size of the enhanced J48 decision tree consisted of 145 nodes and 146 leaves.

3.5 Overall Critique and Current Literature Limitations and Gaps

The relevant studies regarding application layer DDoS detection are notable proposals for detecting malicious DDoS behaviour on the application layer. However, they are not directly applicable in our case. The majority of such studies aim to detect application layer DDoS attacks through monitoring the user web behaviour as shown in [240], [250], [243], [7], [245], [171], [230] and [34]. Hence, they used user web-based behaviour features to differentiate between normal and malicious traffic. This is not fully achievable in IoT networks as there are various devices such as sensors and actuators that no user behaviour is present.

Assessing the relevant literature used to detect DDoS attacks in smart homes, the smart grid and the AMI services, we observe that machine learning techniques are a popular and effective choice as they achieve high detection rates, ranging from 68.8% to 100% TP rate and a FP rate from 1.3 to 80% (although the high 80% FP rate was achieved at the start of the training procedure and then reduced to nearly zero as stated in the study [13]). The literature is also rich and extended beyond smart homes and the smart grid and also has notable studies in other resource constrained networks. Examples included IoT networks, VANETs and WSNs. These studies also achieved remarkable results with TP rates ranging from 72% to 100% and with FP rates ranging from 0.025% to 16.66%.

However, there are some notable gaps in this set of studies. Firstly, with the exceptions of [152], [194], the rest of the studies presented are focused on DDoS attacks that are not on the application layer. Application layer DDoS attacks detection is unarguably harder as they mimic legitimate behaviour convincingly well and might be confused with legitimate traffic events such as flooding to be confused with flash crowd events [241], [92] .

Secondly, there is a clear preference towards deploying machine learning techniques as they have proven to be particularly accurate against DDoS attacks in various layers. However, the majority of them deploy machine learning algorithms that are demanding in computational resources such as CPU power, memory usage and training time. This will make them unsuitable for adoption by devices low in resources in the smart home-smart grid network. Even if they are deployed in devices that are higher in computational resources the model's re-training can be hard to be executed.

Therefore, an IDS deployed for the smart home-smart grid network should be lightweight and flexible from a re-training point of view so it does not interfere with the network's normal functioning. A lightweight solution towards effectively detecting application layer flooding and slow-rate attacks could potentially be the combination of forecasting methods and the usage of Lyapunov exponents as studies presented Section 3.3.3 clearly demonstrated.

Due to the nature of application layer DDoS attacks, it is evident that the proposed IDS solution should monitor closely the present short-term traffic over a set of time-series and not base its knowledge on long-term previous behaviour of the system. Therefore, it should predict what the future short-time-interval traffic, is based on present short-time-interval previous behaviour of the system. Evidently through the set of studies in Section 3.3.3, forecasting methods in combination with Lyapunov exponents is a suitable choice for effective forecasting of short-term future traffic behaviour and its classification to either legitimate or malicious.

However, one particular problem of such methods (without deploying machine learning algorithms as well as the examples in studies [47], [239], [147] and [199]) but also observed in other studies that lighter machine learning choices such as [116] is that they achieve a rather high FP rate when compared with studies that used a more complex but also more resource-demanding approaches. This issue should also be magnified since the nature of application layer DDoS attacks is so similar to legitimate traffic. Therefore, there is clearly need for a multi-layer approach where a more sophisticated second layer is going to be deployed to mainly lower the FP rates but only activated under certain circumstances, so the lightweight nature of the IDS is not diminished.

A lightweight sophisticated and smart technique that manages to improve FP rates and could be deployed on the second layer is logistic regression. Based on the results from [11], [107], [184], [221], [11], [107] and [184], we conclude that logistic regression is an algorithm that achieves a low FP rates under a variety of circumstances and different data. The majority of multi-layer proposals as presented in section 3.4 has a clear motivation of improving TP and FP rates but also conducting multi-attack classification.

Therefore, we have enough evidence to believe that our proposed system shall be a multi-layer approach that on the first-layer shall encompass forecasting algorithms and Lyapunov exponents to achieve a high TP rate and on the second-layer shall encompass logistic regression to ultimately reduce the FP rate achieved and also perform multi-attack classification. Both of these techniques are not high in computational resources unlike other machine learning techniques such neural networks, SVM, decision trees, Bayesian networks and random forests and potentially will achieve good results when combined together to detect application layer flooding and slow-rate DDoS attacks in the smart home-smart grid network.

Chapter 4

ForChaosLR Algorithmic Design

4.1 Overview

As already described in summary of Chapter 3, Section 3.5, our proposed IDS system will consist of two layers. The first layer will be of a combination of simple exponential smoothing forecasting and Lyapunov exponents and the second layer will be of a logistic regression model. Hence, we named the first layer "ForChaos" and the second layer "LR". Combined, the full name of our proposed IDS system is *ForChaosLR multi-layer IDS*.

This chapter describes ForChaosLR multi-layer IDS and how it detects application layer DDoS detection in the smart home-smart grid network. ForChaosLR is a lightweight IDS that will not negatively affect the network's resource-constrained devices. As already explained, in summary of Chapter 3, Section 3.5, ForChaosLR should be lightweight in the following ways:

- A simple exponential smoothing forecasting and a logistic regression model can be re-trained at a much faster rate than a more computationally complex and high-in demand resources machine learning technique such as MLP neural networks, Bayesian networks, random forest and SVM (RBF kernel).
- ForChaos first-layer should be retrained using a smaller in size training set than the mentioned machine learning algorithms.
- ForChaosLR should also not demonstrate significant difference in the resources needed (CPU power, memory usage and execution time) when compared to popular machine learning algorithms such as the ones already mentioned but others as well which have been adopted for the experimental analysis in Chapters 6, 7, 8 and 9.

- To avoid any devices being overstressed from ForChaosLR, only ForChaos first-layer will constantly be monitoring incoming traffic.
- LR second-layer will only be used when ForChaos first-layer detects suspicious behaviour and LR second-layer will need to confirm this malicious nature or dismiss it.
- In the event of an actual DDoS attack being conducted, LR second-layer should further classify the DDoS attack to either flooding or slow-rate.

We strongly believe that it is essential for any IDS proposed for resource-constrained networks, such as the smart home-smart grid network, to encompass the characteristics mentioned. The chapter commences by introducing the relevant features when analysing the network traffic generated in the smart home-smart grid network. Then, the motivation for a multi-layer IDS is argued and supported, making references to the relevant studies introduced in Chapter 3, Section 3.5. Proceeding, the relevant mathematical concepts for the ForChaos first-layer detector is introduced. Then, the pseudocode for ForChaos first-layer is provided. After that, the LR second-layer's mathematical concepts are explained and the relevant pseudocode is provided. LR second-layer's relevant mathematical concepts and pseudocode are explained for both binary and multi-class approaches.

4.2 Feature Selection

Multiple studies that use forecasting techniques against DDoS attacks usually monitor a single feature to detect the possible deviations. Examples include number of packets, number of packets per IP or the packet flags. Some examples presented in Chapter 3 include [71], [218], [148].

However, the adoption of one single feature on the application layer is unlikely to produce satisfactory results since application layer DDoS attacks do not violate any protocol rules, do not produce malformed packets and can be seemingly considered legitimate traffic, as effectively supported by [27], [241] and [92]. Hence, it will be harder to detect application layer DDoS attacks, specifically flooding and slow-rate using only one feature. Furthermore, due to the "legitimacy" of application layer flooding, it could easily be confused with a flash crowd event as studies from [251], [217], [247] and [81] have effectively highlighted.

Hence, we designed a set of multiple features to detect application layer flooding and slow-rate DDoS attacks presented in Table 4.1. Proceeding, we make a brief summary of which studies have inspired the features and also introduce our own new features.

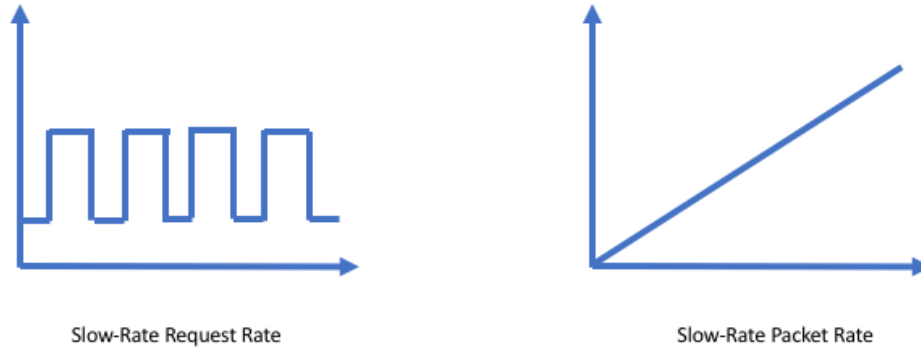


Fig. 4.1 Slow-Rate Packet Behaviour

The majority of IDS studies for DDoS attacks, regardless of the layer deployed, monitor the number of packets. Examples of such studies include [107], [71] and [7]. Therefore we adopted this feature for ForChaosLR multi-layer IDS.

Since we aim to detect both flooding and slow-rate attacks with ForChaosLR, there is need to differentiate between the number of packets sent and the number of requests sent over a time interval. The reason for this distinction is that in a flooding attack, the rate of requests sent is likely to increase exponentially in a short amount of time, therefore the number of packets will also increased exponentially simultaneously.

In contrast, a slow-rate attack's rate of requests will be sporadically increased (in the form of a pulse) as the attackers initiate a large amount of requests. These requests are never completed and as the time passes more are initiated. However, the attackers will maintain these requests alive by sending small in size subsequent packets. Hence, the number of packets will be steadily increased but the rate of increase of requests will be increased in the form of a pulse rate as illustrated in Figure 4.1.

It is important to capture this distinction using these two features both for binary and multi-attack classification. The number of requests as a feature was also used by [230] and [30]. Based on the same assumption, we also adopted the average data rate per interval which was also used by [245] and [176].

Based on the distinction mentioned above, we take a step further and conclude that in a slow-rate attack, the average packet size over a time interval will be significantly decreased

as the majority of packets sent will be smaller than usual. This could have been expected in previous years when the Internet connections were not fast enough. However, in today's world it is highly unlikely that packets are sent in smaller sizes due to bandwidth constraints. Therefore we adopted average packet size per interval as another feature for ForChaosLR multi-layer IDS. The feature of packet size was also used by [199].

In the event of a flooding attack, the number of requests sent over a time interval can be potentially increased exponentially [158], [151]. The same type behaviour could also be potentially demonstrated in a flash crowd event. In the event of a slow-rate attack, the number of requests sent over a time interval will demonstrate a pulse-rate behaviour.

To distinguish flooding from flash crowd but also accurately identify slow-rate DDoS, we strongly believe that the features of average time between two subsequent requests, average time between two subsequent responses and the average time between a server response and the next request generated would be a valuable distinguishing factor. This assumption is supported by the authors in [175] and [199], who effectively supported this. Based on this assumption, we adopted the former feature and also proposed two additional features, the average time between a server response and the next request sent and the average time between two subsequent responses.

Finally, we proposed a new feature "Parallel Requests Number" over a time interval. This feature aims to detect slow-rate attacks more efficiently. Slow-rate DDoS attacks maintain a large number of requests that are yet to be completed. The majority of legitimate requests being either completed in one packet or with a small number of subsequent ones. Hence, if this number is steadily increased over a period of time, it is highly likely that a large number of opened parallel requests indicates a slow-rate DDoS attack. The final list of features is presented in Table 4.1.

1. Requests Number: In a flooding scenario, the number of requests over a period of time is likely to be much larger compared to legitimate traffic request rates. In a slow-rate attack scenario, the number of requests over two consecutive periods of time will have a large difference. Since slow-rate opens connections to send requests after an amount of time, there will be a pattern of low number of requests, then high number of requests, then low and so on.
2. Packets Number: In a flooding attack, the number of packets on the application layer is rapidly increased over a short period of time. In a slow-rate attack, the number of packets is lower due to the packets being sent slowly, just before a request can be rejected. Hence, the request stays alive but very few packets are sent.

Table 4.1 ForChaosLR Features Table

<i>Feature Name</i>	<i>Description</i>
Requests Number	Total number of requests in a time-series interval
Packets Number	Total number of packets in a time-series interval
Average Data Rate	Average data rate in Megabits in a time-series interval
Average Packet Size	Average packet size in a time-series interval
Average Time Between Requests	Average time between two requests in a time-series interval
Average Time Between Response & Request	Average time between the response and the first requests encountered in a time-series interval
Average Time Between Responses	Average time between two responses in a time-series interval
Parallel Requests Number	Total number of parallel requests in a time-series interval

3. Average Data Rate: In a flooding attack, the data rate on the application layer is rapidly increased over a short period of time. In a slow-rate attack, the data rate is lower due to the slow data rate of the malicious requests initiated.
4. Average Packet Size: In a flooding attack, the average packet size is decreased due to the requests being simply a get request that features no payload. In a slow-rate attack, the average size packet is decreased since the series of connections send small incomplete packets over the attack period of time.
5. Average Time between Requests: In a flooding attack, the average time between each request is significantly decreased due to an outburst of requests being sent over a short period of time simultaneously. In a slow-rate attack, the average time between each request is decreased when compared to the same variable in a legitimate scenario. Instead, there are spikes of the number of requests being sent. When the attack is at the stage of initiating new requests, the rate of requests is increased and the time between each of them is decreased. When the attack is at the stage of maintaining the connections open, normal subsequent packets are sent, therefore, the rate of the requests is less than the previous stage and the time between them is increased. Then, when new requests are initiated the rate is again increased and their time decreased.

6. Average Time between Response and Request: In a flooding attack, the average time between a response and the next request is significantly decreased. In a slow-rate attack, the average time between a response and the next request the slow-rate attack malicious request is also increased, as the server is not able to normally respond to requests since they are never completed.
7. Average Time between Responses: In a flooding attack, the average time between two consecutive responses is significantly decreased since a burst of requests is being sent to the server, therefore a great number of responses must be generated. In a slow-rate attack, the same feature's value is also increased since the slow-rate attack's requests are initiated but never completed.
8. Parallel Requests: In a flooding attack, the average concurrent requests is decreased since the attackers send multiple requests that are rapidly completed. In a slow-rate attack, many requests are initiated and stay open over a period of time and the average number of parallel open requests is large.

4.3 Motivation for a multi-layer approach IDS

Network traffic state can fluctuate randomly and unpredictably due to legitimate reasons. Hence, application layer flooding and slow-rate DDoS attacks are harder to be distinguished from normal legitimate behaviour as they do not violate any protocol rules, mimic legitimate burst traffic (e.g. flash crowd) and can be stealthy in the saturation of resources [241], [92]. Hence, an IDS should be effectively trained in such a way that can detect both attacks and distinguish between them and legitimate incoming traffic and any potential spikes that might be present.

One way of detecting such attacks is to monitor their behaviour with respect to time. Time could potentially assist in differentiating between malicious and legitimate behaviour. The proposed IDS system must be able to detect changes in behaviour of the network by monitoring its behaviour over a set of time-intervals. Due to the nature of the attacks, it is evident that the IDS system should monitor closely the present short-term traffic and not base its knowledge on long-term previous seen behaviour of the system. The reason is due to traffic rate being such a volatile entity with respect to time.

Therefore, we should predict what the future short-time-interval traffic is based on present or short-time-interval previous observed behaviour of the system instead of classifying the present network traffic monitoring based on long-term previous behaviour of the system. It

is important to avoid incorrectly classifying some type of behaviour as malicious, simply because it is not similar to the history of the network.

Inevitably, there will always be some error between a prediction and its actual value. The objective is to effectively assess the error, so we can understand when the error is large enough to indicate that there is a DDoS attack present. There are multiple approaches in assessing an error. Examples include statistical approaches, distance metrics and entropy as demonstrated in studies [26], [116], [71], [169], [176] and [199].

For our research, we chose Lyapunov exponents, due to their suitability for our problem to be solved. Hence, in the lightweight first layer of detection we deploy ForChaos first-layer detection algorithm. ForChaos uses simple exponential smoothing forecasting to make short-term predictions and identify potentially malicious behaviour through the construction of Lyapunov exponents for every time-series interval. To assess whether there is an indication of a DDoS attack in the network, Lyapunov exponents are used to indicate whether chaotic behaviour is present.

As described in the Chapter 3, Section 3.5, it is important for the deployed IDS in the smart home-smart grid network to be lightweight so it does not negatively affect the resource-constrained IoT devices' functioning. However, as previous studies indicated, including [116], [256], [169] and [148], lightweight IDS solutions can be effective in detecting malicious behaviour but achieve high FP rates.

A potential issue observed, especially with forecasting techniques, is that they are "trained" with only normal behaviour and create a series of predictions. Whatever falls out of this set of expected predictions, is classified as malicious. This is likely to cause issues in correctly classifying incoming traffic.

Therefore, there is need for a more sophisticated second layer of detection. The second layer should be trained with both legitimate and malicious behaviour. In that way, it can have a clear view of what is considered legitimate and malicious as well as their possible variations. The second layer of ForChaosLR multi-layer IDS is named LR second-layer and should be able to classify what kind of behaviour is present in the system, whether that is a flooding or a slow-rate DDoS attack.

LR second-layer aims to reduce the FP rate that is expected from ForChaos to generate. It also aims to classify on the type of attack, if present. Therefore, every time ForChaos generates an alert for malicious traffic present, the sophisticated LR second-layer should analyse the traffic in further. Through its analysis, it should decide on whether the alert is indeed an attack or a false positive event. In that way, LR second-layer is not constantly in use and the lightweight approach is not dismissed.

ForChaos first-layer is likely to require a short amount of observations to start producing meaningful results as observed from previous studies such as [71], [169], [176], [218], [148], [199] and [47] which used forecasting algorithms. In addition, forecasting algorithms only require legitimate traffic observations to construct their model. Hence, ForChaos first-layer will be re-trained in a short amount of time compared to more complex machine learning algorithms such as Bayesian networks, MLP neural networks, random forest and SVM (RBF kernel) algorithms. In addition, ForChaos first-layer will be retrained using a smaller in size training set than the machine learning algorithms mentioned. As a result, ForChaos first-layer will be re-trained more frequently and using less resources.

In Figure 4.2, we present the architecture of the ForChaosLR Multi-Layer IDS architecture with regards to binary classification (attack or normal).

Both detection algorithms consist of a "training phase" to construct their algorithmic detection models. ForChaos first-line detection algorithm only requires real-time normal traffic for its training. LR second-layer detection algorithm needs offline legitimate and malicious traffic for its training. Both detection algorithms will perform feature extraction using the same features to build their models.

During evaluation, the monitoring of unknown traffic is conducted by ForChaos first-layer. ForChaos monitors incoming traffic by constructing predictions about the future traffic state. If the total error (average error of the errors for each feature monitored) generated between prediction and actual network traffic state is large enough to produce a positive Lyapunov exponent, then the "suspicious" traffic is forwarded to LR second-layer for further investigation. If the incoming traffic generates a negative Lyapunov exponent (or very rarely a value of zero), then the incoming traffic is treated as legitimate and it is not forwarder to LR second-layer.

In the event of LR second-layer receiving forwarder traffic, it uses its constructed logistic regression model to decide on whether the suspicious network traffic is indeed a DDoS attack. In practice, if the probability generated (based on the logistic regression model) is less than 0.5, then the forwarded traffic is classified a DDoS attack and an alert is generated. If the probability is higher than 0.5, then the forwarded traffic is legitimate.

In Figure 4.3, we present the architecture of the ForChaosLR Multi-Layer IDS architecture with regards to multi-class classification (either flooding, slow-rate, normal or flooding, HULK, slowhttptest, slowloris, goldeneye, normal).

Both detection algorithms consist of a "training phase" to construct their algorithmic detection models. ForChaos first-line detection algorithm only requires real-time normal traffic for its training.

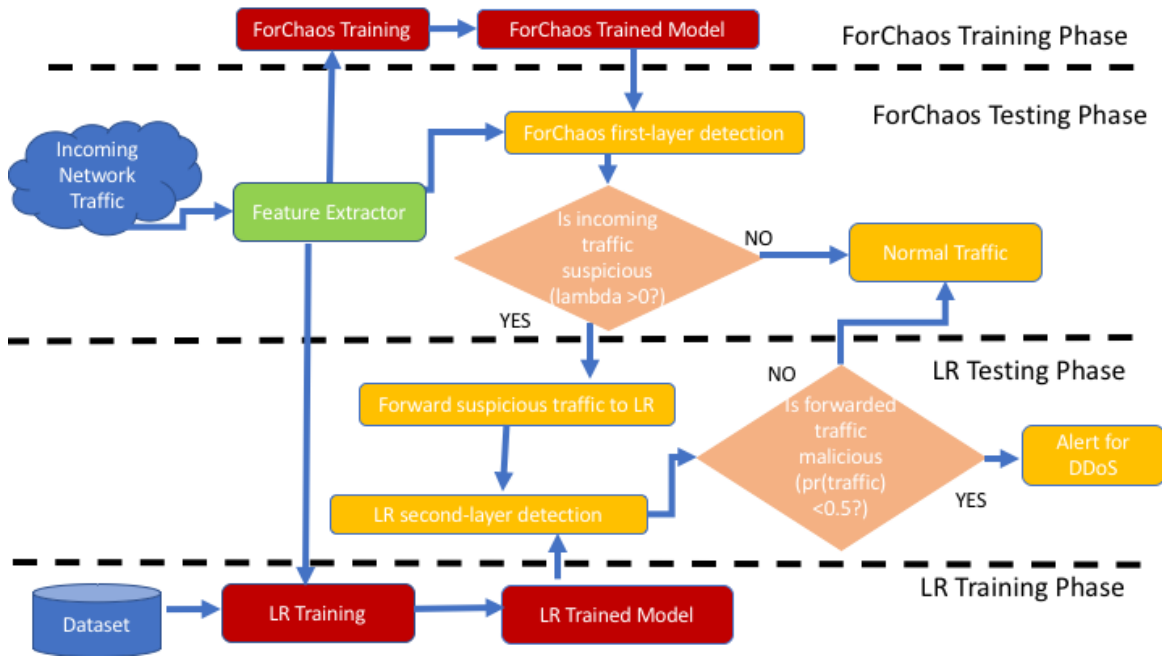


Fig. 4.2 ForChaosLR Multi-Layer IDS Architecture

LR second-layer detection algorithm needs offline legitimate and malicious traffic (for each specific attack category defined) for its training. In contrast to binary classification, LR second-layer constructs multiple logistic regression models, equal to the total number of labels present in the training set.

Both detection algorithms will perform feature extraction using the same features to build their models.

During evaluation, the monitoring of unknown traffic is conducted by ForChaos first-layer. ForChaos monitors incoming traffic by constructing predictions about the future traffic state. If the total error (average error of the errors for each feature monitored) generated between prediction and actual network traffic state is large enough to produce a positive Lyapunov exponent, then the "suspicious" traffic is forwarded to LR second-layer for further investigation as well as specific attack classification. If the incoming traffic generates a negative Lyapunov exponent (or very rarely a value of zero), then the incoming traffic is treated as legitimate and it is not forwarder to LR second-layer.

In the event of LR second-layer receiving forwarder traffic, it uses its constructed logistic regression models to decide on whether the suspicious network traffic is indeed a DDoS attack and if it is, which specific DDoS attack is. In practise, each logistic regression model uses the data forwarded to calculate the probability of the data belonging to a specific class. In the end, the logistic regression model that achieves the maximum probability has the data classified with its label.

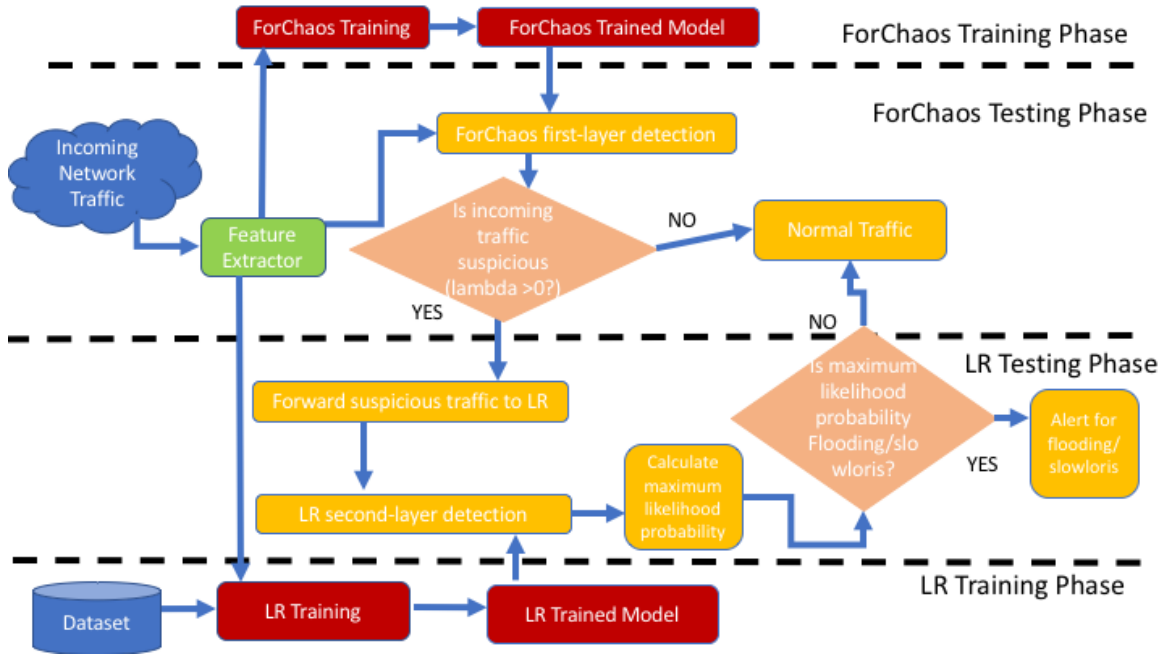


Fig. 4.3 ForChaosLR Multi-Layer IDS Architecture (multi-class)

4.4 ForChaos First-Layer Modelling

4.4.1 Forecasting to Predict Future Network Behaviour of the System

Based on the arguments presented, we choose to exclude simple forecasting techniques such as moving average (MA) and weighted moving average (WMA). The main reason for our choice is that in order for a forecasting model to make accurate forecasts, it should have a balance between past forecasting and observations. Instead, we choose simple exponential smoothing forecasting.

Our argument is supported by [71], where the authors conducted an evaluation study between MA, WMA and simple exponential smoothing forecasting. It concludes that simple exponential smoothing forecasting is the most effective forecasting algorithm as opposed to the other two against DDoS attacks. In the relevant literature, the most popular forecasting methods that use exponential smoothing forecasting, besides the already mentioned, are double exponential smoothing forecasting and triple exponential smoothing forecasting. Simple exponential smoothing forecasting has no means of considering trend if there is one present in past observations while double exponential smoothing forecasting considers the potential trend present in data. Triple exponential smoothing forecasting also considers trend in data and seasonality, something that neither double exponential smoothing forecasting or simple exponential smoothing forecasting do.

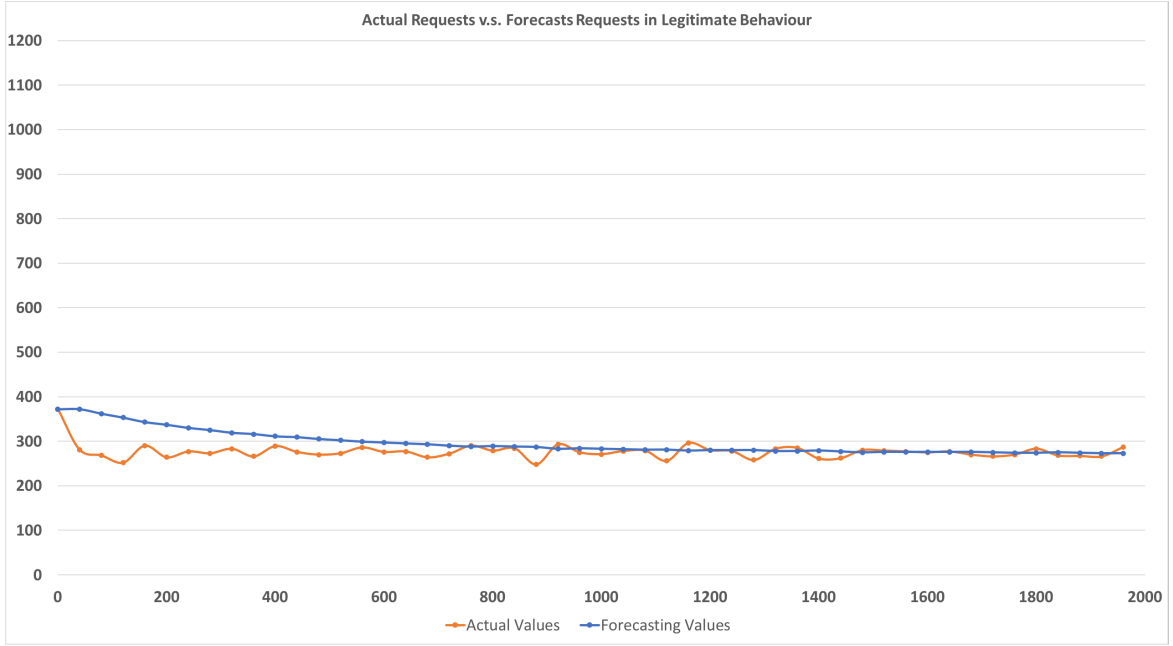


Fig. 4.4 Simple Exponential Smoothing Forecasting in legitimate traffic

In an observed time series consisting of network traffic observations for every feature in the features table 4.1, let A_i be the state of traffic at time i . Hence, we get a time-series of:

$$A_1, A_2, A_3, \dots, A_i, \dots, A_n \quad (4.1)$$

For every state of traffic A_i we have a corresponding forecast F_i . F_i is defined as the forecast at time i . A_{i-1} is the previous actual observation, F_{i-1} is the previous forecast and α is defined as the smoothing constant. Smoothing constant α can take the values $0 \leq \alpha \leq 1$. The equation of F_i is presented firstly presented in Equation 2.1. However, for practical reasons, we re-present the Equation below in 4.2. An example of how simple exponential smoothing forecasting works on a DDoS attack event is presented in Figure 4.4.

$$F_i = \alpha \cdot A_{i-1} + (1 - \alpha) \cdot F_{i-1} \quad (4.2)$$

The choice of α value plays an important role in having a balance between the forecast and actual observation. By having α leading towards 1, more weight is given to the actual observation. If α is leading towards 0, more weight is given to the past observations. Since we have a set of features at every time interval i , a forecast for each of the features is made based on history.

4.4.2 Chaos Theory for Error Analysis

It is inevitable that every forecasting system produces a series of errors between their prediction and the observed values. The error at every time interval i is defined by Equation 4.3.

$$e_i = |F_i - A_i| \quad (4.3)$$

At every time interval, we construct a vector of eight values (as the features are eight) based on the network traffic obtained. Proceeding, we calculate a vector of eight different corresponding prediction values. Then, a vector of eight errors is constructed by subtracting each actual value from its corresponding forecasting one.

We define the total error from all the features shown in Equation 4.4 where e_i is the error value for every of the n features at time i . In ForChaos $n = 8$, as 8 is the total number of features used.

$$E_{total} = \frac{1}{n} \sum_{i=1}^n e_i \quad (4.4)$$

Temporary spikes in the traffic state can be observed without the presence of a DDoS attack for various reasons. Some examples include the sudden legitimate increase in generated traffic from existing devices or new devices entering the network. In both examples, the network traffic state is suddenly legitimately increased. Hence, the error generated from the forecast and the actual traffic state might be temporarily increased for a relative short period of time but eventually settle down to its natural distribution values.

In contrast, in the presence of a DDoS attack, the forecasts for the network traffic state will continuously diverge from the actual traffic state, leading to their exponential separation. As a result, the overall prediction of future short-term behaviour unstable and highly inaccurate. As a consequence, the error generated will be increased exponentially with time. A representation of such behaviour is presented in 4.5.

To assess this error in further, we use the Lyapunov exponent, as explained in Chapter 2, Section 2.5.2. We treat the error generated using chaos theory instead of stochastic processes. The literature defines that stochastic processes describe systems and phenomena that appear to vary in a random manner [58], [211], [210]. Therefore, stochastic systems treat events as random. If we consider two successive realisations of a random process, we will get two different sequences, even if the initial state is the same. On the other hand, "chaotic behaviour" denotes the fact that trajectories arising from initial data which are arbitrarily close grow apart exponentially. Chaos theory takes into consideration that a small difference

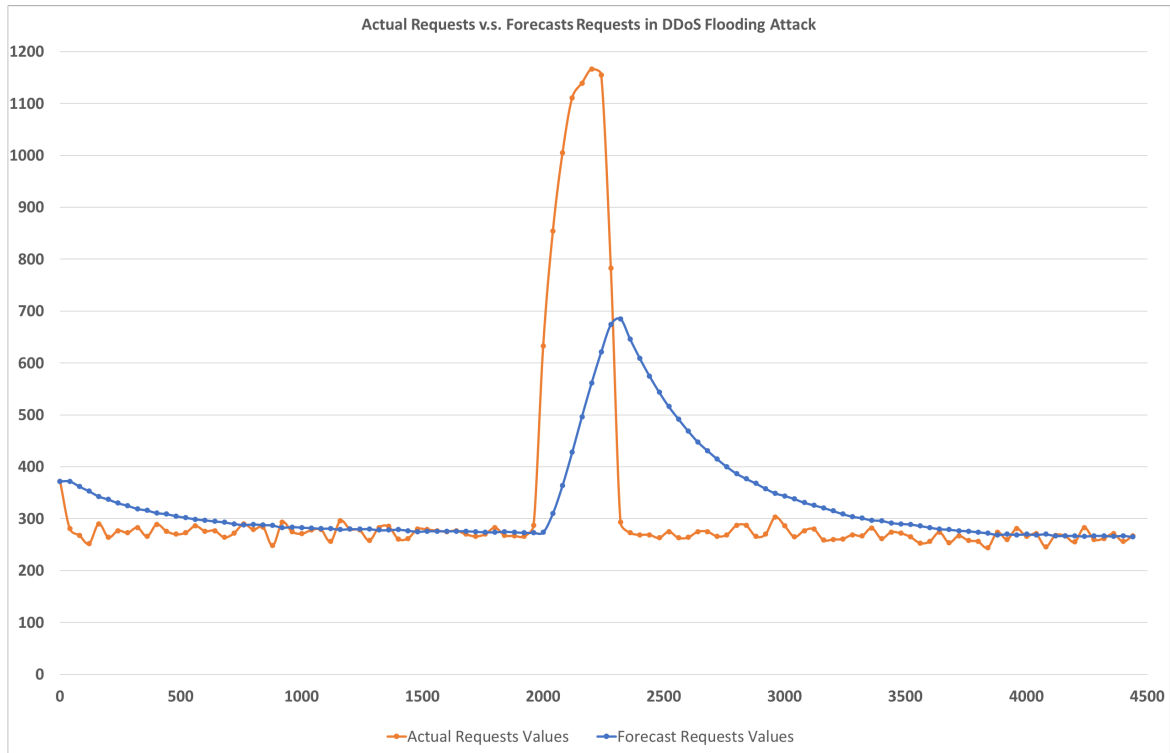


Fig. 4.5 Simple Exponential Smoothing Forecasting in DDoS attack

in the current state can result in significant differences in its future state. This was very important for our research as we wanted to model such behaviour. As stated, we believe the value of the error between the prediction model and the actual network traffic behaviour should be analysed using Lyapunov exponents.

In our opinion, the error value (rate of separation between these two trajectories) should not be treated as a random event. Lyapunov exponents measure the rate of separation (aka error) between the two trajectories (prediction and actual network traffic). If this rate is positive, then the system's behaviour is chaotic and therefore, we would need to investigate it in further as there is an underlying condition on why this is the case (possibility of a DDoS attack) as the rate of separation keeps increasing exponentially and does not settle. From a prediction point of view, this means that the forecasting model is unable to produce any forecasts about the network behaviour that are relatively accurate. Furthermore, for an error to be flagged as unacceptable and forwarded for further investigation, a threshold has to be defined as studies in [71], [233]. This threshold however can be biased since it will be decided after some sort of “training phase”. On the other hand, when Lyapunov exponents are used no threshold has to be decided and bias regarding this part of the algorithm can

be avoided. These properties of Lyapunov exponents and chaos theory are what ultimately made us choosing to proceed with this approach.

Based on the explanations regarding chaos theory, we make the following assumptions regarding our system.

1. We assume that in the presence of legitimate traffic the two trajectories (forecast and actual values for traffic state) converge and therefore have small distance from each other, thus the distance δZ (error) between them is relatively small.
2. We also assume that the actual traffic trajectory can diverge, but likely to (not always) adjust due to burst legitimate traffic entering the network (such as flash crowd events).
3. Finally, we assume that if the actual traffic trajectory diverges exponentially but does not adjust, it is due to a DDoS attack present in the system.

$$\lambda_t = \begin{cases} > 0, & \text{DDoS attack.} \\ \leq 0, & \text{Legitimate traffic.} \end{cases} \quad (4.5)$$

Hence, Equation 2.10 is changed to Equation 4.6, where δZ is transformed to error e . In general, δZ denotes the separation of two trajectories. In our case, this separation is the error between the actual and the forecast values.

$$\lambda_t = \frac{1}{t} \cdot \ln \left| \frac{e_{total_t}}{e_{total_0}} \right| \quad (4.6)$$

4.4.3 ForChaos First-Layer Training

In this section, we explain the training procedure of ForChaos first-layer detection algorithm. Evidently, a factor that directly affects the accuracy of simple exponential smoothing forecasting model is α . Its optimal value is decided during training phase. The last factor that must be considered in the time window size w . The time window size denotes how frequently a new forecast will be calculated. For example, if the window size is set to 40 seconds, then a forecast will be calculated at the 40th, 80th, 120th, 160th second and so on. To assess all these different combinations between α and window size w , we used the root mean square error (RMSE) in Equation 4.7. The training time is symbolised by T and denotes the final training time required from ForChaos first-layer to be able to produce accurate forecasts.

$$RMSE = \sqrt{\frac{\sum_{t=1}^T (e_{total_t})^2}{\frac{T}{w}}} \quad (4.7)$$

Choosing a window size w is an important step regarding ForChaos first-layer for two main reasons.

- Simple exponential smoothing makes a prediction based on past observations over a series of time so it is directly dependent on it.
- Application layer flooding and slow-rate DDoS attacks are very similar to legitimate traffic. Both flooding and flash crowd consist of a sudden burst of traffic observed in the network over a very short period of time. Hence, it is difficult to separate legitimate from malicious traffic. Also, slow-rate attacks are harder to detect as they exploit the time a server waiting time for a request to be completed.

The time window sizes of forecasting IDS studies in Chapter 3 ranged from 60 second to 300 seconds [71], [147], [176], [218], [148]. The majority of the studies used ICT traffic. In the AMI network of the smart grid, AMI data exchange from smart meters ranges from 1 to 120 minutes according to [204]. However, in our study we consider smart home and smart grid traffic so a large window size is unlikely to be efficient since the traffic interchange between devices and networks is quicker.

Based on the arguments presented above, we conducted a series of experiments using different windows sizes. Initially we evaluated the first series of experiments using 30, 40 and 60 seconds window sizes. After a series of experiments we analysed the results and decided that the 40-second time window size is the most optimal to use for ForChaos first-layer in smart home-smart grid traffic. The term "optimal window size" denotes the window size value which produced the lowest RMSE during training.

More details regarding the evaluation of the window sizes are presented in Chapter 6, Section 6.5.

A summary of the ForChaos first-layer detection algorithm training is summarised in algorithm 1. At each window size, the values for the eight features are calculated based on the incoming traffic A_t . A forecast is made for each feature, according to line 1 in algorithm 1. This forecast is compared with the actual value at every feature and an error is calculated. The total error is calculated as the summation of errors for all eight features divided by 8 in line 9. During training, the optimal combination of α and time window size w is assessed by calculating the RMSE at the end of every training procedure with a specific α and w at line 10. The RMSE value is compared with the minimum value in line 11. If it is less, then the minimum value takes the RMSE value and the corresponding α and w are stored in α_{min} and w_{min} respectively. Value α and w are incremented by 0.1 and 10 respectively. If α reaches 0.9 then t and α are re-initialised to 0 and 0.1 respectively and the loop is re-executed. The training is terminated when all the values of w are evaluated, each with all the values from

0.1 to 0.9. The optimal window size w and α are returned as well as the corresponding set of previous forecasts F_t .

Algorithm 1 ForChaos first-layer Training

```

1: Input: Training Time  $T$ , Traffic State  $A_t$ , Set of 8 Features  $f = f_1, f_2, \dots, f_8$ 
2: Output:  $w_{min}, \alpha_{min}, F_{min}$ 
3:  $\alpha_{min} = 0, w_{min} \approx \infty, e_{total} = 0, \alpha = 0.1, w = 10, RMSE = 0, min = 0, F_0 = 0$ 
4: while  $w < 60$  do
5:   for  $t = 0$  to  $T/w$  do
6:     for  $f = 1$  to  $8$  do
7:        $F_t \leftarrow \alpha A_{t-1} + (1 - \alpha) F_{t-1}$ 
8:        $e_t \leftarrow |F_t - A_t|$ 
9:        $e_{total_t} \leftarrow e_{total_t} + \frac{1}{8} \sum_{i=1}^n e_i$ 
10:       $RMSE \leftarrow RMSE + \sqrt{\frac{\sum_{i=1}^{T/w} (e_{total_t})^2}{T/w}}$ 
11:    if  $RMSE < min$  then
12:       $min \leftarrow rmse.$ 
13:       $\alpha_{min} \leftarrow \alpha.$ 
14:       $w_{min} \leftarrow w.$ 
15:     $\alpha \leftarrow \alpha + 0.1$ 
16:     $t \leftarrow 1$ 
17:    if  $(\alpha > 0.9)$  then
18:       $\alpha \leftarrow 0.1$ 
19:       $w \leftarrow w + 10$ 
20: return  $F_t, w_{min}, \alpha_{min}$ 

```

4.5 LR second-layer Modelling

As described in Chapter 2, Section 2.5.8, logistic regression aims to create the best approximating function between a dependent variable, in our case whether there is a DDoS attack or not, and the independent variables, in our case the features chosen. In the next section, we explain the procedure followed by LR second-layer both for binary and multi-class classification.

4.5.1 LR second-layer Training for Binary Classification

Through logistic regression, we calculate a probability of an instance i belonging to a class ($y = 1$) or not belonging to a class ($y = 0$). Specifically in our case, the binary probability of an incoming traffic state to be a DDoS attack or not. Our aim is to calculate the optimal

coefficients of logistic regression's sigmoid function. This is the main objective of training. The sigmoid function of the logistic regression is given in Equation 4.8.

$$Pr(y) = \frac{1}{1 + e^{-(g(x))}} \quad (4.8)$$

where $g(x)$ is denoted by Equation 4.9, c_i are the optimal coefficients and x_i is the corresponding value for every feature x . The term i takes values from 1 to 8 since 8 is the total number of features used.

$$g(x) = c_0 + \sum_{i=1}^8 c_i \cdot x_i \quad (4.9)$$

To effectively construct the logistic regression model for LR second-layer, we followed the procedure described in Chapter 2, Section 2.5.8. To minimise and potentially prevent overfitting, we use a value called ridge. For effective algorithmic construction during training phase, cost functions are involved. The general idea is for the cost function to be minimised during training. This results into a more accurate and successful algorithmic model as the literature suggests. In general, one approach to decrease the cost function consists of increasing the number of features used. Increasing the number of features is likely to help the algorithm fit the training data better and therefore, decrease the cost function.

Generally, to decrease the cost function, we increase the number of features in our model. As we keep on increasing the features in model, model starts fitting the training data set well and cost function value starts decreasing. However, the downfall of increasing the set of features is that the algorithmic model's polynomial order is increased and it can potentially lead to overfitting. To overcome this issue, a ridge value (small bias) is used. Ridge value quantifies the overfitting of data by calculating the magnitude of the coefficients of the algorithmic model used [100], [153], [101].

Ridge value is used in the cross entropy Equation 2.17 as an additional penalty value multiplied by the square of $m \cdot (k - 1)$ where m is the number of features and k is the number of classes. Thus, the new cross entropy function is transformed into the Equation 4.10:

$$NewCross = \left[-\frac{1}{n} \sum_{i=1}^n a_i \cdot \ln(p_i) + (1 - a_i) \cdot \ln(1 - p_i) \right] + ridge \cdot (m \cdot (k - 1))^2 \quad (4.10)$$

4.5.2 LR second-layer training for Multi-class Classification

In the previous section, we described how logistic regression is trained when each training sample i belongs to a particular class ($y = 1$) or it belongs to that particular class ($y = 0$), in

our case whether it is a DDoS attack or not. Based on this, we can expand and generalise the logistic regression model so that an instance i can belong to many classes with $y \in \{1, \dots, C\}$ where C denotes the total number of classes. In our case, we expand the model to have the following labels: flooding, slow-rate and normal. Beyond the general flooding and slow-rate categories, during the experimental evaluation presented in Chapter 8, we categorised malicious behaviour in further by grouping the relevant instances to the specific type of attack, namely: flooding, HULK, Slowloris, HTTPTest and Goldeneye [92].

The major difference between binary and multi-class logistic regression is that instead of the sigmoid function, the softmax function is used as given in Equation 4.11. Softmax function calculates the probability of an instance x belonging to class j , where $y = j$ given the inputs for each feature and the respective weights of class j along with its bias, as presented in Equation 4.12. This procedure is repeated for every class and in the end the argmax is taken and the appropriate class label is assigned to instance x . The training procedure is the same with binary logistic regression since the gradient descent algorithm is used to minimise the cross entropy function. However, instead of sigmoid, softmax function is used.

Softmax function is used in multinomial logistic regression. We proceed with describing how it operates below:

The softmax function receives an input vector k which consists of m real numbers and then normalises it into a probability distribution consisting of m probabilities to the exponentials of the m real numbers (input). After applying softmax, each number will be in the $[0, 1]$ interval so all negative or higher than 1 numbers will be mapped to the $[0, 1]$ range. In addition, all the numbers will add up to 1. The class with the highest probability achieved, is assigned to the input vector instance [235].

$$Pr(z)_i = \frac{e^{z_i}}{\sum_{j=0}^C e^{z_j}} \quad (4.11)$$

where $g(x)$ is denoted by Equation 4.12 and c_i are the optimal coefficients.

$$z = c_0 + \sum_{i=1}^n c_i \cdot x_i \quad (4.12)$$

4.6 ForChaos First-Layer and LR Second-Layer Detection Model

When training is completed, ForChaosLR multi-layer IDS can be used to detect malicious behaviour in the smart home-smart grid network in real time. Let incoming traffic be Θ .

Incoming traffic Θ is divided in time-series according to the optimal window size w chosen. Therefore, we get a time-series of incoming network traffic states as shown in 4.13.

$$\Theta = \theta_1, \theta_2, \theta_3, \theta_4, \dots, \theta_n \quad (4.13)$$

From each network traffic time-series at θ_t , the set of eight values for f features are extracted and passed to ForChaosLR multi-layer IDS. For each feature a forecast F is calculated based on Equation 4.2. Every forecasting method, including simple exponential smoothing forecasting, is bound to generate an error rate, which is the difference between the forecasting and the actual value. Proceeding, we calculate the Lyapunov exponent using the Equation 4.14.

$$\lambda_t = \frac{1}{t} \ln \left| \frac{e_{total_t}}{e_0} \right| \quad (4.14)$$

In Equation 4.14, e_0 is the first total error calculated. A positive Lyapunov exponent indicates chaotic behaviour at time instant t . This means the forecast prediction F_t demonstrates large distance to the observed value A_t . Hence, the incoming network traffic behaviour observed is chaotic and unstable. This means that the changes in the incoming network traffic observed were not expected based on previous behaviour. As a result, the change could be due to an attack. On the contrary, a negative Lyapunov exponent shows that the error is not chaotic because the difference between the forecast and the actual observation is small.

$$\lambda_t = \begin{cases} > 0, & \text{Chaotic behaviour and indication of DDoS attack.} \\ \leq 0, & \text{Normal behaviour and no indication of DDoS attack.} \end{cases} \quad (4.15)$$

When the Lyapunov exponent is positive, the suspicious traffic needs to be processed in further to decide if there is a DDoS attack. Specifically, for every positive Lyapunov exponent λ_i , where $i \in [0, \infty]$ the eight features are considered in the logistic regression model with window size of 40 seconds. Based on the sigmoid function result, the probability of a specific network traffic being classified as either malicious or legitimate is defined as:

$$Pr(x) = \begin{cases} < 0.5, & \text{Alert.} \\ \geq 0.5, & \text{No Alert.} \end{cases} \quad (4.16)$$

The full procedural functioning of ForChaosLR multi-layer IDS is presented in Algorithm 2. The input traffic θ_t at every time-series t is received as an input. For every feature, a forecast is made and compared with the actual value at time-series t . For each feature, an error is generated. The total-error for all of eight features is calculated and used as input to

calculate the Lyapunov exponent for that time-series t . If the Lyapunov exponent is positive, the logistic regression model is applied. The extracted features are forwarded as an input to the logistic regression detection model. If the probability $Pr(x)$ is less than 0.5, then an alert of a DDoS attack is made as shown in Algorithm 2 .

Algorithm 2 ForChaosLR Detection Algorithm

```

1: Input: Set of 8 Features  $F = f_1, f_2, \dots, f_8$ , Time Series  $t$ ,  $\alpha$ , Window Size  $w$  Traffic State  $\theta_t$ , Sigmoid Function  $S$ , Sigmoid Function Window Size  $w_s$ ,  $\text{argmax} = 0$ 
2: Output: alert, no-alert
3:  $e_{total} = 0, F_0 = 0$ 
4: for  $t = 1$  to  $n$  do
5:   for  $f = 1$  to  $8$  do
6:      $F_t \leftarrow \alpha A_{t-1} + (1 - \alpha) F_{t-1}$ 
7:      $e_t \leftarrow |F_t - A_t|$ 
8:      $e_{total_t} \leftarrow e_{total_t} + \frac{1}{8} \sum_{i=1}^n e_i$ 
9:      $\lambda_t \leftarrow \frac{1}{t} \ln \left| \frac{e_{total_t}}{e_0} \right|$ 
10:    if  $\lambda_t > 0$  then
11:      for  $f = 1$  to  $8$  do
12:         $Pr(X) \leftarrow \frac{1}{1 + e^{-(g(x))}}$ 
13:        if  $Pr(X) < 0.5$  then
14:          return alert
15:    else
16:      return no – alert

```

4.7 ForChaos First-Layer and LR Second-Layer Multi-Attack Detection Model

The detection procedure of ForChaosLR multi-layer IDS when it is used as a multi-attack detection model is identical to the binary version up until the step which ForChaos first-layer forwards the suspicious network traffic to LR second-layer. When LR second-layer is trained using a multi-class approach, it does not only detect application layer DDoS attacks but also classifies it to either flooding or slow-rate. This procedure is presented in lines 11-19 in Algorithm 3. For every suspicious and potential DDoS traffic forwarded by ForChaos first-layer, LR second-layer processes it for every one of the three classes (flooding, slow-rate, normal) (line 11) by substituting the data on every formula for every class (line 13). If the probability for the current class is higher than the previous set probability (line 14) then the new argmax is the newly calculated argmax of the current class, j in the algorithm (line 15),

and class y is set to value j (line 16). After the incoming data was substituted in all three classes, a final if statement is executed where it is checked whether the assigned class with the highest probability is either flooding or slow-rate (line 18). If it is either of the two then an alert is generated. The procedure is identical to the case of the DDoS attacks being classified in a specific attack type, specifically flooding, slowloris, HTTPTest, hulk and goldeneye. The only difference is the number of probabilities to calculate for each of the classes required.

Algorithm 3 ForChaosLR multi-class detection algorithm

Input: Set of 8 Features $F = f_1, f_2, \dots, f_8$, Time Series t , α , Window Size w Traffic State θ_t , Sigmoid Function S , Sigmoid Function Window Size w_s , $j \in C = \text{flooding, slow-rate, instance}$

Output: alert, no-alert

```

3:  $e_{total} = 0, F_0 = 0$ 
   for  $t = 1$  to  $n$  do
     for  $f = 1$  to  $8$  do
       6:  $F_t \leftarrow \alpha A_{t-1} + (1 - \alpha) F_{t-1}$ 
           $e_t \leftarrow |F_t - A_t|$ 
           $e_{total_t} \leftarrow e_{total_t} + \frac{1}{8} \sum_{i=1}^n e_i$ 
       9:  $\lambda_t \leftarrow \frac{1}{t} \ln \left| \frac{e_{total_t}}{e_0} \right|$ 
          if  $\lambda_t > 0$  then
            for  $j = 1$  to  $C$  do
              12:  $Pr(y = j | g(x)) = \frac{e^{g(x)_j}}{1 + \sum_{j=0}^C e^{g(x)_j}}$ 
                  if  $argmax < argmax g(x)_j$  then
                     $argmax = argmax g(x)_j$ 
              15:  $y = j$ 
                  if  $y = \text{flooding or slowloris}$  then
                    return alert
              18: else
                    return no – alert

```

4.8 Summary

In this chapter, we presented ForChaosLR multi-layer IDS's overall functioning. Its mathematical operation was explained in depth, as well as how the different components interact with each other. In detail, the relevant mathematical concepts used in ForChaos first-layer detection algorithm are explained. Proceeding, the relevant mathematical concepts for the LR second-layer are presented and explained how they were applied in DDoS detection. Also,

the relevant pseudocode was provided for both layers as well as for ForChaosLR multi-layer IDS as a whole.

Chapter 5

Methodology of Experiments

5.1 Overview

This chapter introduces the relevant tools and hardware used for the experiments described in Chapters 6, 7, 8 and 9. These tools/applications will constitute the testbed for evaluating ForChaosLR multi-layer IDS. ForChaosLR multi-layer IDS should be a lightweight IDS that can be frequently re-trained without needing a significant amount of resources. It should also achieve a high accuracy in detecting flooding and slow-rate application layer DDoS attacks with a minimal FP rate. Hence, two approaches are to be followed:

- An accuracy evaluation approach, where we examine ForChaosLR multi-layer IDS's performance in correctly classifying incoming traffic. This evaluation is conducted in Chapters 6, 7 and 8.
- An evaluation on Raspberry Pi 3 model B+, where we examine ForChaosLR multi-layer IDS's memory usage, CPU power and training/testing time requirements. This evaluation is conducted in Chapter 9.

As stated, ForChaosLR multi-layer IDS is a lightweight IDS proposed solution for resource-constrained and IoT devices for the smart home-smart grid network. Hence, it was essential for our proposed system to be tested on an actual resource-constrained device to confirm that our objectives defined in Chapter 1, were successfully met. Furthermore, these two experimental approaches were adopted for a set of machine learning (ML) algorithms to compare their achieved results with the equivalent ones of ForChaosLR multi-layer IDS. As any other study, our research, design, implementation and evaluation of ForChaosLR multi-layer IDS has a number of limitations. We briefly describe these below.

1. There is no dedicated dataset containing smart home and smart grid data available (both application layer flooding and slow-rate DDoS attacks and normal) so we had to construct our own dataset by means of simulations.
2. We did not have access to an appropriate network and therefore could not collect real network data.
3. The purchase of smart home-smart grid network devices for the evaluation of For-ChaosLR multi-layer IDS is not feasible.

Based on the limitations presented above and the objectives defined in Chapter 1, we made a constructive plan for the experimental testbed. We define the objectives of the experimental testbed as the following:

1. A simulation of the smart home-smart grid network along with the relevant network protocols used should be implemented using a network simulator based on the official standards and guidelines available.
 - Specifically, the traffic generated in our simulation that followed a Poisson distribution [43], [163].
 - Regarding the smart grid standards and guidelines, to simulate this part, we used the standard messages/signal times as defined by [204]. Smart meters, whether they are located in smart homes or in other domains on the smart grid report smart meter pricing the typical interval is between 15 minutes to 1 hour. Demand/response and load shedding signals could be reported as frequently every 2 seconds. Therefore, we varied a the transmission of such signals from 8 seconds to 15 minutes.
 - Proceeding, we analysed the 2018 and 2019 smart home datasets which contained real-world smart home traffic. In their smart home network, the authors deployed 30 different smart home devices, including smart cameras, blood pressure meters, NEST protect smoke alarm, smart sleep sensors, smart bulb, android/iPhone devices, laptops and so on. The maximum amount of packets generated per second was 58.7. Our simulation which consisted of 10 nodes generated an average of 18.5-22.1 packets per second. More information regarding the relevant datasets and our simulation can be found in Chapter 5, Section 5.4.4.
2. Through the smart home-smart grid network simulation, both legitimate and DDoS traffic should be generated based on real-world cases or other network traffic simulations available. More information can be found in Chapter 5, Section 5.4.5.

3. Through the smart home-smart grid network simulation enough data should be collected, so a dataset is effectively constructed. This dataset should be sufficient to train both ForChaosLR and the ML algorithms.
4. Sufficient implementation for both ForChaosLR and the ML algorithms should be developed.
5. Both ForChaosLR and ML algorithms should be tested on a low-resource constrained device. The device chosen is Raspberry Pi 3 model B+, which features a Cortex-A53 64-bit CPU, 4 cores, a 1.4 GHz CPU clock, and 1GB of RAM memory.

Based on the objectives defined, we proceed with presenting all the relevant tools/applications and datasets needed to meet these objectives.

5.2 Datasets used for ForChaosLR multi-layer IDS Evaluation

To effectively evaluate ForChaosLR's accuracy across different datasets we used multiple datasets for different purposes. These datasets were either generated through our smart home-smart grid network simulation or acquired from credible universities which collected traffic from dedicated physical networks.

All datasets raw data are in .pcap files. Feature extraction was performed on raw data, so meaningful training and test sets could be constructed to train and evaluate ForChaosLR multi-layer IDS and the machine learning algorithms. Proceeding, we describe the datasets used for the experimental analysis and what each dataset's purpose is. A summary of all the datasets used is in Table 5.1.

5.2.1 NS-3 Binary Datasets

Initially, we had to create a smart home-smart grid network dataset. Through it, we aimed to evaluate ForChaosLR multi-layer IDS by generating enough data through our smart home-smart network simulated in NS-3 simulator [75]. More details regarding NS-3 are presented in Section 5.3. The first step would be to evaluate on whether ForChaosLR could differentiate between legitimate and malicious DDoS traffic. Hence we followed a binary dataset construction approach. In this subsection, we briefly describe the procedure followed to create the NS-3 binary training and testing sets for ForChaosLR and the ML algorithms.

Table 5.1 Summary of datasets

<i>Dataset name</i>	<i>Dataset Type</i>	<i>Network type</i>	<i>Attack instances number</i>	<i>Legitimate instances number</i>
NS-3 Binary Training Set	binary	smart home-smart grid	3757	3230
NS-3 Binary Testing Set	binary	smart home- smart grid	2073	10762
ISCXIDS2012 Training Set	binary	ICT network	2548	3712
ISCXIDS2012 Testing Set	binary	ICT network	94	2426
CICIDS2017 Training Set	binary	ICT network	1862	1119
CICIDS2017 Testing Set	binary	ICT network	130	295
IEEE TMC 2018 Training Set	binary	smart home	6784 (Bot-IoT dataset)	5877
IEEE TMC 2018 Testing Set	binary	smart home	n/a	1973
ACM SOSR 2019 Training Set	binary	smart home	3392 (Bot-IoT dataset)	6280
ACM SOSR 2019 Testing Set	binary	smart home	n/a	1263
NS-3 Multi-class Training Set	multi-class	smart home-smart grid	n/a	3228
NS-3 Multi-class Testing Set	multi-class	smart home- smart grid	1896(flooding) 1868(slow-rate)	3228
NS-3 Multi-class Testing Set	multi-class	smart home- smart grid	991(flooding) 967(slow-rate)	10323
CICIDS2017 Multi-class Single Category Training Set	multi-class	ICT network	840(flooding) 1055(slow-rate)	440
CICIDS2017 Multi-class Single Category Testing Set	multi-class	ICT network	30(flooding) 101(slow-rate)	378
CICIDS2017 Multi-class Multi-Category class Training Set	multi-class	ICT network	840(flooding) 924(slowloris) 462(slowhttpstest) 720(hulk) 806(goldeneye)	880
CICIDS2017 Multi-class Multi-Category class Testing Set	multi-class	ICT network	30(flooding) 30(slowloris) 31(slowhttpstest) 24(hulk) 13(goldeneye)	378

NS-3 Binary Training Set

This dataset was constructed to train ForChaosLR and the ML algorithms used in the experimental evaluation chapters (Chapters 6,7,8 and 9). As already stated in Chapter 4 Section 4.7, ForChaosLR consists of two layers that function in a different way. Therefore, the training process followed needed to be differentiated. ForChaos first-layer's training procedure involved the usage of 25 normal instances. The detailed procedure of ForChaos first-layer training is explained extensively in Chapter 6, Section 6.5. For LR second-layer and the ML algorithms' training, involved of both legitimate and DDoS instances. To avoid any bias and/or dominant class issues we used the SMOTE algorithm [1].

Imbalanced data is a real-world problem in classification. Most of the times, data is bound to demonstrate some sort of imbalance. In practice, that means that one or more classes in the dataset do not contain equal proportion of data compared to other(s). Heavily imbalanced classes can negatively impact the machine learning algorithms' performance thus, making them produce unsatisfactory results. In more detail, the machine learning algorithm tends to classify incoming data as instances belonging to the majority class(es). This occurs because not enough data is available for smaller in size classes, so the algorithm faces issues in constructing an effective model that is able to understand such data.

SMOTE solves this issue by creating new synthetic instances that belong to the minority class(es) based on the existing minority class(es) instances. More details regarding how SMOTE algorithm works can be found in Chapter 2, Section 2.6. Hence, the final version of the training set for both LR second-layer and the ML algorithms consisted of 3757 attack and 3230 legitimate instances.

NS-3 Binary Testing Set

As stated in Chapter 4, Section 4.7, LR second-layer only evaluates traffic forwarded from ForChaos first-layer. Therefore, we created two testing sets: one for the ForChaosLR and one for the ML algorithms. The ML testing set consisted of 2073 attack and 10762 legitimate instances. The LR second layer testing set consisted of 2037 attack and 772 legitimate instances, as the rest of them were handled from ForChaos first-layer.

5.2.2 Real World Binary Datasets

Despite constructing a dataset using the simulated smart home-smart grid network, we decided it was important to also evaluate both ForChaosLR and ML algorithms in datasets constructed in physical network environments. This approach was adopted for a number of reasons.

Initially, we aimed to examine whether the ForChaosLR and ML algorithms results acquired from the NS-3 binary dataset were reflected on real-world datasets that contained application layer DDoS attacks. Through this comparison we would eventually confirm two assumptions.

- The first assumption is that our NS-3 dataset being realistic enough since the results gained are similar in rates.
- The second assumption is that ForChaosLR multi-layer IDS is able to correctly classify application layer DDoS attacks and legitimate traffic in traditional ICT networks, despite being a lightweight detection system designed specifically for low-resource constrained and IoT networks such as the smart home-smart grid network.

Proceeding, we describe the procedure followed to create the real-world training and testing sets for ForChaosLR and the ML algorithms.

ISCXIDS 2012 Training and Testing Sets

ISCXIDS 2012 was constructed by the Canadian Institute for Cyber Security from the University of New Brunswick [5]. The dataset contains realistic network traffic and diverse intrusion scenarios. Real traces were used and analyzed to create profiles for agents. These agents are responsible for generating realistic traffic with examples being HTTP, SMTP, SSH, IMAP, POP3, and FTP traffic. In order for such behaviour to be correctly simulated by agents, human users' behaviour was studied and the agents were programmed accordingly to mimic such behaviour. To simulate malicious behaviour, various multi-stage and diverse intrusion attack scenarios were simulated. The attacks were significantly varied in frequency, size and complexity.

ISCXIDS 2012 dataset contains 7 days of normal and malicious traffic. We used a total of 4 days which consisted of 3 days of legitimate traffic and 1 day of application layer flooding DDoS traffic. The rest of the dataset contained other types of cyberattacks and intrusions that were not relevant to our study. Below, we proceed with constructing the training and testing sets.

For training ForChaos first-layer, we used a total of 25 legitimate instances. For the training set of LR second-layer and the ML algorithms the procedure required an additional series of steps. Since there was only approximately 1 hour of application layer flooding DDoS present, it was evident that we had to construct synthetic flooding DDoS instances, hence we used the SMOTE algorithm [1]. The final training set version consisted of 2548 attack and 3712 legitimate instances. For testing the ML algorithms, the total of instances

consisted of 94 attack and 2426 legitimate instances. For testing LR second-layer, 94 attack and 2150 legitimate instance were used, since ForChaos first-layer correctly classified the remaining 276 as legitimate.

CICIDS 2017 Training and Testing Sets

CICIDS 2017 was also constructed by the Canadian Institute for Cyber Security from the University of New Brunswick [105]. It contains real-world traffic simulated in a real-world environment. Real traces were analyzed so agent profiles could be created. Once again these agents generated realistic traffic such as HTTP, HTTPS, FTP, SSH and email protocols. The dataset consists of 5 days of traffic in total. During the 2 two days, no malicious activity occurred. The rest of them contained various simulated attacks. Examples included as web-based, brute force, DoS, DDoS, infiltration, heart-bleed, scanning, and application layer flooding and slow-rate DDoS attacks. For our dataset construction, we used the 2 days of legitimate activity and 2 days which consisted of the application layer slow-rate and flooding DDoS attacks.

For training ForChaos first-layer we used a total of 25 legitimate instances. For training LR second-layer and the ML algorithms, the procedure required an additional series of steps. Since the traffic consisting of the DDoS attacks was not sufficient for both training and testing, it was evident that we had to construct synthetic flooding DDoS instances, hence we used the SMOTE algorithm. The final training set version consisted of 1862 attack and 1119 legitimate instances. For the testing set of ML algorithms, 130 attack and 295 legitimate instances were used. For testing ForChaosLR multi-layer, 130 attack and 291 legitimate instances were used, as ForChaos first-layer classified the remaining ones correctly.

Smart Home Datasets

A real smart home-smart grid network is unlikely to be identical to a traditional ICT network or even a simulated smart home-smart grid in terms of overall behaviour. To evaluate in further and ultimately prove that ForChaosLR multi-layer IDS's results are accurate and representative enough of what a real-world case would be, we used the IEEE TMC 2018 [206] and ACM SOSR 2019 [4] datasets from the UNSW University of Sydney.

The IEEE TMC 2018 dataset contains 20 days of legitimate traffic from a real world smart home environment containing 30 different IoT and smart devices such as smart cameras, smart bulbs, amazon echo, blood pressure meter, NEST Protection alarm and others. The complete list can be found in [206]. The ACM SOSR 2019 dataset contains both legitimate and DDoS malicious traffic on the same smart home environment as the IEEE TMC 2018

dataset. However, these DDoS are not on the application layer and could not be used for our experiments.

In general, our main goal was to evaluate if ForChaosLR multi-layer IDS is able to identify realistic incoming smart home traffic as legitimate, since the evaluation of whether it can detect application layer DDoS traffic should be satisfied through the ISCXIDS 2012 and CICIDS 2017 datasets. ForChaos first-layer does not require any malicious traffic to construct its algorithmic model however, LR second-layer needs both legitimate and malicious instances for its training phase. Hence, we used the application layer DDoS attack traffic from the The BoT-IoT dataset [170].

For training ForChaos first-layer we used a total of 25 normal instances. The IEEE TMC 2018 training set of LR second-layer consisted of 6784 attack and 5852 legitimate instances. The IEEE TMC 2018 testing set consisted of 1973 legitimate instances. The ACM SOSR 2019 training set of LR second-layer consisted of 3392 attack and 6280 legitimate instances. The ACM SOSR 2019 testing set consisted of 1263 legitimate instances.

5.2.3 NS-3 and CICIDS 2017 Multi-Class Datasets

The first step in evaluating a proposed IDS is to ultimately test whether it can correctly classify normal and malicious traffic, in our case application layer flooding and slow-rate DDoS attacks. IDS systems can also categorise malicious traffic to a specific attack. Such IDS studies are likely to follow a multi-layer IDS approach. This approach consists of the malicious traffic being detected (typically in the first layer) and then forwarded to a more complex model to be mapped to a specific type of attack (typically in the second layer).

ForChaosLR's multi-layer nature primarily consists of improving the overall accuracy from the first to the second layer since it is a primary objective. However, it is always important for an IDS to perform attack type classification, especially for mitigation purposes. Therefore, we also constructed multi-class versions of the NS-3 and CICIDS 2017 datasets to evaluate ForChaosLR multi-layer IDS's ability in performing multi-class classification. In the next sections, we outline how the re-construction of datasets already used occurred. The training and testing sets for ForChaos first-layer remained unchanged, as only LR second layer is responsible for mapping the malicious behaviour to specific attacks.

NS-3 Multi-class Training Set

The procedure to transform the NS-3 binary training set to multi-class was quite simple, as the attack instances were renamed to either flooding or slow-rate depending on what they represented. SMOTE algorithm [1] was once again used to avoid any bias and training

imbalance due to minority classes. The total number of flooding instances was 1896, of slow-rate was 1868 and of legitimate was 3228.

NS-3 Multi-class Testing Set

The procedure to transform the NS-3 binary testing set to multi-class was identical to the training set transformation. Once again, the attack instances were renamed to either flooding or slow-rate depending on what they represented. The total number of flooding instances was 991, of slow-rate was 967 and of legitimate was 10323.

CICIDS 2017 Multi-Class Single Category Training Set

The procedure to transform the CICIDS 2017 binary training set to multi-class was quite simple, as the attack instances were renamed to either flooding or slow-rate depending on what they represented. SMOTE algorithm [1] was once again used to avoid any bias and training imbalance due to minority classes. The total number of flooding instances was 840 of slow-rate was 1055 and of legitimate was 440.

CICIDS2017 Multi-Class Single Category Testing Set

The procedure to transform the CICIDS 2017 binary testing set to multi-class was the same as the training set transformation. Once again, the attack instances were renamed to either flooding or slow-rate depending on what they represented. The total number of of flooding instances was 30, of slow-rate was 101 and of legitimate was 378.

CICIDS2017 Multi-Class Multi-Category Training Set

CICIDS 2017 dataset actually consists of multiple types of slow-rate and flooding attacks such as slowloris, slowhttptest goldeneye, hulk and LOIC flooding [92]. We decided to evaluate ForchaosLR's and other ML algorithms' potential ability to even further classify the attacks to a specific type and not just a category. As a result, the number of classes was changed to normal, flooding, hulk, slowhttptest, slowloris and goldeneye and the instances were renamed accordingly. SMOTE algorithm [1] was once again used to avoid any bias and training imbalance due to minority classes. The total number of flooding instances was 840, of slowloris was 924, of slowhttptest was 462, of hulk was 720, of goldeneye was 806 and of normal was 880.

CICIDS2017 Multi-Class Multi-Category Testing Set

For the CICIDS 2017 testing set the same procedure as for the training set was followed. Once again, the attack instances were renamed based on what specific attack they represented. The total number of flooding instances was 30, of slowloris was 32, of slowhttptest was 31, of hulk was 24, of goldeneye was 13 and of normal was 378.

5.3 Network Simulator 3 (NS-3) Simulator

Real-world system networks are complex and contain many different interconnected components that interact with each other. Analysing such systems and conducting experiments on them can be difficult and practically infeasible. Thus, using discrete network simulators is the best alternative. Proceeding, an investigation of the open-source simulators was conducted to choose the most suitable network simulator. Below the most popular network simulators are briefly described.

- Omnet++ was made available to the public in 1997 [226]. Omnet++ is an open source discrete simulation framework. Although primarily used for network simulations, it can also be used for multiprocessors and distributed systems modelling and evaluation of complex systems.
- Network Simulator 2 (NS-2) is an open source discrete network simulator released in 1996 [93]. It can simulate different protocols and topologies, both wired and wireless. Simulation scripts are written in OTcl language, which is an extension of the Tcl scripting language. OTcl is used for constructing the network topology. The source code of ns-2 is written in C++. NS-2 runs on GNU/Linux, FreeBSD, Solaris, Mac OS X and Windows versions that support Cygwin. Network animator is used for representing the simulated network graphically.
- Network Simulator 3 (NS-3) began being developed in 2006 as a replacement of NS-2, with the first version released in 2008 [75]. It is also an open-source discrete network simulator. NS-3 is not backwards compatible with NS-2. The reason was because it was built from scratch, therefore it was not an update to NS-2. OTcl was completely abandoned and everything was rebuilt in C++. There is also scripting capability with Python. It also has a range of both wired and wireless models such as WiFi, WiMAX and LTE. NS3 also supports emulation and tapbridge modes of operation for NS-3 to interact with the real world.

After researching on the available network simulators we chose NS-3 as our main simulator for the smart home-smart grid network simulation. We chose NS-3 mainly because it provides very good support in socket programming and client/server architectures. In addition, through NS-3, the entire network traffic generated can be captured and saved in pcap files. These can later be used for training and testing of ForChaosLR multi-layer IDS. Through NS-3, objectives 1-4 will be satisfied. Below, we briefly describe the most relevant terms used in NS-3 and their definitions since we are going to frequently refer to them throughout the chapter.

- **NODE**: In the real world, any device that can be connected to the Internet is called a host. In NS-3 a host is called a node and represents an abstract computer device entity. It is handled by the class Node which is written entirely in C++. A node object can have behaviour involving protocol stacks, applications and peripheral cards.
- **APPLICATION**: An application in the real-world of computer systems denotes a piece of software that either helps the device function and manage its hardware components (operating system) or helps the user achieve a task. Since NS-3 is a network simulator, the idea of an application is mainly represented as generating traffic in the form of a thread. This is achieved by class Application which is entirely written in C++. Some popular examples of specialised application libraries in NS-3 are UdpEchoClientApplication and UdpEchoServerApplication.
- **CHANNEL**: In physical computer networks, a channel connects a computing device to a network, a network with another network or a set of networks, or directly two computing devices together. In NS-3 the same concept is present through the Channel class written in C++. A channel object is attached to a node so it can be "connected to the Internet". Channel class can connect nodes together or networks together or a mixture. A channel object can be either wired or wireless.
- **NET DEVICE** : In the real world, to connect a computer to the Internet except the (wireless or wired) channel, a network interface card is also needed along with a software driver to effectively manage it. In NS-3, the same concept is implemented through the Device class, written in C++. Through the Device class, a node in NS-3 can be connected to multiple channels.

5.4 Smart Home-Smart Grid Network Simulation

The smart home-smart grid network consists of multiple and different communication channels, technologies and standards. Hence, it was essential to research the available literature, to identify all business-communication use cases. In the next subsections we present the most critical and relevant nodes in the smart home-smart grid network. Next, we give a detailed description of the network protocols used in the smart home-smart grid network. Based on this knowledge, we present the implementation procedure and details of the smart home-smart grid network simulation in NS-3, including the nodes set up, protocols simulation, generation of legitimate traffic and DDoS attacks.

5.4.1 Smart Home-Smart Grid Network Critical Nodes

- **Energy Services Interface (ESI):** ESI is located on the smart home network but also on other commercial and/or industrial premises. It acts as the information management gateway through which the Customer domain interacts with various services provided by the smart grid [224]. ESI is responsible for demand response and provides the customer with useful energy usage information. Through it, the remote control of devices and demand response programs are enabled, the monitoring of distributed energy resources such as wind turbines belonging to premises can also be achieved and the forwarding of consumption data to the neighborhood collection points (if it acts as a meter) and plug in electric vehicles/ plug in electric hybrid vehicles (PEV/PHEV) charging is conducted. It is also responsible for load shedding handling and remote load control.
- **Smart meter:** An electronic device located in residential, commercial and industrial premises. It is responsible for recording consumption of electric energy in frequent intervals and for communicating that information back to the utility for monitoring and billing [224].
- **Smart appliance:** Smart appliances are devices that evolved from being traditional isolated products that serve a home owner's needs to smart devices. They are mainly located in residential premises but can also be found in commercial and/or industrial buildings.
- **AMI Headend:** AMI Headend is located at the edge of the AMI infrastructure. It receives all the incoming data to forward it to the respective utility servers. It may perform some limited data validation before forwarding the data.

- Demand Response Management System (DRMS): DRMS is responsible for sending demand/response signals to the consumers. These signals can be either of emergency type or early signals or as part of the agreed consumption between the customer and the utility.
- Load Management System (LMS): LMS is responsible for sending emergency signals to the customers. These signals are used to shut off energy consuming entities and help the smart grid continue performing normally.
- Customer Information System (CIS): CIS is the interface between the customer direct interactions with the smart grid's services. Through it the customer can enlist and manage its smart appliances, sign up to different energy consumption programmes and so on.
- Outage Management System (OMS) helps the smart grid staff identify the reason why an outage occurred and perform all the necessary actions to restore power or other services.
- Data Management System (DMS) is responsible for ensuring the smart grid communications particularly from the smart grid side are safe and secured.

5.4.2 Smart Home-Smart Grid Network Protocols

Although NS-3 consists of a wide range of protocols, there were no implementations of application protocols available. Therefore, we had to simulate the IoT protocols we described in Chapter 2 Section 2.2.4. Below, we give a brief explanation of our approach to simulating the IoT protocols, focusing on the study of each protocol's header and how this could be simulated in NS-3.

HTTP Protocol Header

A client sends a request packet with a request header. The general HTTP request message consists of a request line (e.g., GET /images/myCat.jpeg HTTP/1.1), the relevant request header fields (e.g., Accept-Language: en), an empty line and sometimes a message body depending on the type of the request. The request line and other header fields must each end with <CR><LF> which is equivalent to an empty line and no whitespaces. In our implementation of the HTTP header in NS-3 we implemented the header values as String fields. The appropriate setters were implemented to set their values when creating an HTTP header object, depending on the type of packet. When the server receives the packet from the

client, it uses getter functions to deassemble the header, check the header values and then respond accordingly, by sending the correct type of packer with the correct header values. The size of all the header values had no particular limit as we cannot have an indication of how long some of the headers are going to be some such as the URL path. Although there is already an unofficial HTTP implementation available online [195], we chose to study it and be inspired by the work of it, but eventually implement our own implementation.

Since many of the IoT devices are low in resources such as memory and processing power, lightweight alternative versions of traditional application layer protocols, such as HTTP, had to be developed. Below we give a brief description of the protocols and their implementation in NS-3.

CoAP Protocol Header

As stated in Chapter 2, CoAP was inspired by the HTTP protocol and was intended to be a lighter version of the former, specifically target for the IoT devices. For every request header there are some mandatory header fields and some optional. The mandatory fields are Version, Type, Option Count, Code and Message ID. The optional fields are Tokens, Options and Payload. As already stated, no application layer protocols are officially implemented in NS-3 so we implemented our own libraries. As already stated, IoT protocols are lightweight network protocols so the low-resources devices are not overstressed. Thus, their header fields are small in size. Below we give a brief explanation of what each header field is responsible for, its size and what values it can take.

- Version (Ver): 2-bit unsigned integer. Specifies the version of the CoAP protocol. For the time being, the value of this header field is always specified to 1 as it is the first version of the protocol. In the future, the value is going to be incremented.
- Type (T): 2-bit unsigned integer. Specifies the type of the message. The values this header field can take are Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3).
- Option Count (OC): 4-bit unsigned integer. Specifies the number of options (in the option field) present that are after the header. If the value for this header field is set to 0, then there are no options present in the packet.
- Code (Code): 8-bit unsigned integer. Specifies the request method if the packet is sent by a client or the response method if the packet is sent by the server. The client's requests code vary from 1-10 and the server's from 4-0-255. The client's most popular request codes are 1 for GET, 2 for POST, 3 for PUT and 4 for DELETE. The server's

most popular response codes are 41 for CREATED, 42 for DELETED, 43 for VALID, 44 for CHANGED, 45 for NOT FOUND, 46 for NOT AUTHORISED and so on. It is evident that the request and response type of the CoAP header are very similar to the HTTP header.

- **Message ID (ID):** 16-bit unsigned integer. Checks for message duplication and also assists in matching Acknowledgement/Reset messages to Confirmable/Non-Confirmable messages.

MQTT Header

MQTT is another popular lightweight protocol for IoT devices. Similar to CoAP, it contains mandatory and optional header fields in an attempt to make the packets as light as possible. Once again, for every request header there are some mandatory header fields and some optional. The mandatory fields include the Message Type, the Dup Flag, The Quality of Service (QoS), message retain and remaining length. The optional fields include the variable length header and the payload. Below we give a brief explanation of what each header field is responsible for, its size and what values it can take.

- **Message Type:** 4-bit unsigned integer. It specifies the type of the message to be sent by a code. In detail, this header field can take values from 0-15. 0 denotes Reserved, 1 denotes Connect, 2 denotes ConnaAck, 3 denotes Publish and so on.
- **Dup:** 1-bit unsigned integer. It specifies a duplicate flag that indicates the receiver of the message that the message was already received. It can only take the value of 0 or 1.
- **QoS:** 2-bit unsigned integer. It specifies the level of delivery. 0 denotes that the message will be delivery at mostly once. 1 denotes that the message will be delivered at least once. 2 denotes that the message will be delivered exactly once.
- **Retain:** 1-bit unsigned integer. It specifies on whether the server should retain the last PUBLISH message and send it to new subscriptions.
- **Remaining Length:** 8-bits unsigned integer. It specifies what the remaining bytes in the message are about such as payload and variable length size.

XMPP Header

XMPP's structure is a bit different than the other two IoT protocols implemented. The reason is because it is combined with XML. Therefore, it follows a similar approach to XML. XMPP

allows a stanza (message) to consist of multiple child elements, each of them having specific content. The mandatory fields though are the from, to, stream and version. The 'from' is the client name sending the packet, the 'to' indicates the receiver of the packet, the stream denotes the actual URL requesting and the version denotes the version of the XMPP used.

IoT Protocols Headers Implementations

As already stated, no application layer protocols are officially implemented in NS-3 so we implemented our own libraries. In our implementation of the CoAP and MQTT headers in NS-3 we had to implement our protocol header fields as integers, inspired by the work in [208] for the CoAP header. However, the smallest field type size is 8 bits. Therefore, we used the shift-bit methods in C to construct the fields. The appropriate setters were implemented to set their values when creating an CoAP or MQTT header object, depending on the type of packet. When the server receives the packet from the client, it uses getter functions to deassemble the header, check the header values and then respond accordingly, by sending the correct type of packer with the correct header values. The size of all the header values had no particular limit as we cannot have an indication of how long some of the headers will be (e.g. URL path). For the XMPP header, all of the fields described are in String format and not in bits.

DLMS/COSEM

In smart grid operations, DLMS/COSEM (Device Language Message Specification/Companion Specification for Energy Metering) protocol belongs to the IEC 62056 set of standards for electricity metering data exchange as well as control and management [119]. DLMS/COSEM protocol is used for automatic meter reading. Specifically, DLMS models the communication between different entities and COSEM creates the rules based on the relevant standards for meter data exchange. A DLMS/COSEM implementation was already available by [119], so we directly installed the component into NS-3 to use it in our simulations.

5.4.3 Smart Home-Smart Grid Network Simulation in NS-3

Various communication architectures were proposed for the successful information transfer from the smart home to the AMI Headend of the smart grid and vice versa. Mainly, the ESI and/or the smart meter act as the two access points for the smart home and transfer relevant information to the smart grid from the various smart appliances installed at homes.

The ESI has two communication channels to send the information to the AMI services, either the Internet or through a data concentrator unit which acts as a gateway to a neighbour-

hood of smart homes. For practical reasons and due to smart meter's constrained capabilities and potential physical tampering, the ESI can act as the single gateway and most of the communication occurs through the Internet. In order to simulate the smart home-smart grid network, we define a set of assumptions for the simulation. These are briefly described below.

1. In the smart home network, the gateway should be the energy services interface (ESI).
2. In the industrial/commercial/reliability part of the smart grid network, there should be multiple devices from potentially different networks with all corresponding to one gateway.
3. In the AMI network of the smart grid network, the gateway should be the AMI headend.
4. Both the smart home network and the smart grid industrial/commercial/reliability part of the smart grid network should communicate with the AMI part of the smart grid through the AMI headend, but they cannot communicate with each other.
5. All the nodes in the smart home-smart grid network should be static, with no mobility.
6. All the nodes in the smart home and the industrial/commercial/reliability part of the smart grid should be able to communicate with other nodes in their perspective network as well as the base station deployed on each network.
7. All nodes in the smart home-smart grid network should communicate with other nodes using specific protocols, depending on their type and the network deployed. As a result, every node should have their communication specifications associated to the TCP/IP protocol stack across all layers.
8. Every node generating traffic should have a degree of randomness in creating and sending packets. Furthermore, a degree of randomness should be used in the size of the packet, the payload and the commands on it.

Based on the assumptions defined above, we proceed with describing the simulation of the smart home-smart grid network by describing the network topology, protocols used and parameters defined for the traffic generation, including both legitimate and DDoS attacks.

At the smart home side, 11 nodes were simulated. 10 of them represented IoT devices deployed in a smart home, such as smart appliances (e.g. smart fridges, smart oven, smart washing machines), smart sensors (e.g. motion sensors, flood/fire detection sensors, temperature/humidity sensors) and various IoT devices (e.g. light systems, smart door locks, smart thermostats, smart speakers) and the eleventh one represented the ESI.

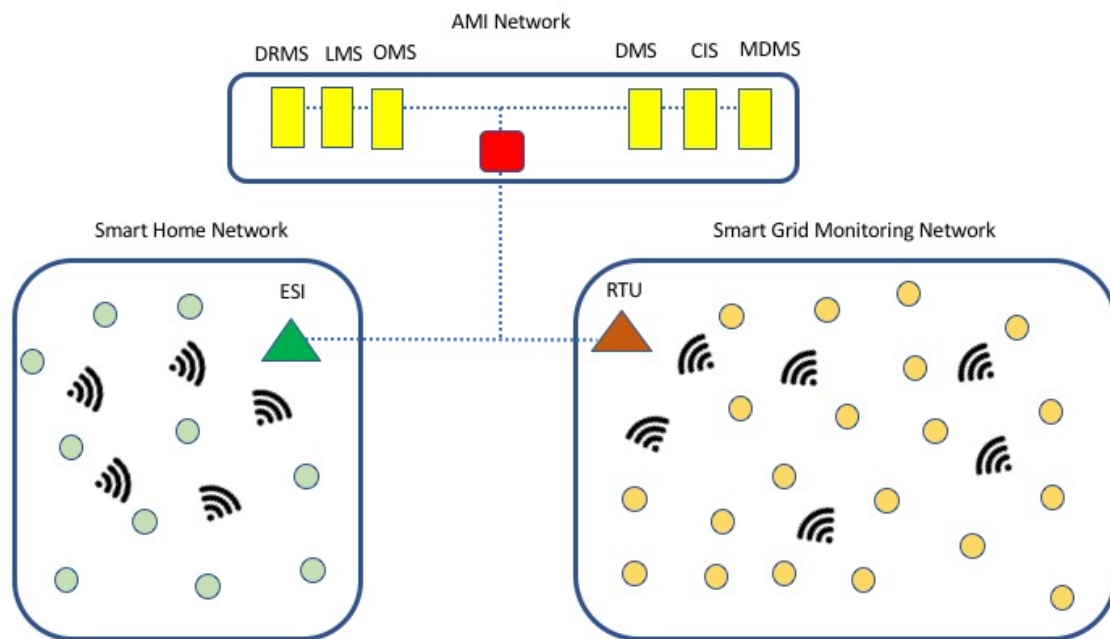


Fig. 5.1 Smart Home-Smart Grid network simulation in NS-3

All nodes were static, with no mobility. In the physical layer, all the connections were wireless and the 802.11 protocol was used. The structure was set to ad-hoc, using ad-hoc on demand distance vector routing (AODV) routing protocol. IPv4 was used in the network layer, and TCP/UDP in the transport layer. In the application layer a wide variety of application layer protocols were simulated including CoAP, MQTT, XMPP and HTTP.

At the industrial/commercial/reliability part of the smart grid network, 21 nodes were simulated. 20 of them represented devices that are typically deployed in a smart grid network such as smart meters, phasor measurement units, relays, intelligent electronic devices, digital fault detectors, renewable energy generators and sensors. The final node represented the gateway that forwards data and signals to the AMI Headend.

All nodes were static, with no mobility. In the physical layer, all the connections were wireless and the 802.11 protocol was used. The structure was set to ad hoc, using AODV routing protocol. In the network layer IPv4 was used and in the transport layer both UDP and TCP were used. In the application layer the application layer used was DLMS/COSEM.

At the Advanced Metering Infrastructure (AMI) part of the smart grid network, 6 nodes were simulated. 5 of them represented the AMI's utilities, MDMS, CIS, DRMS, LMS, OMS and DMS. The final node represented the AMI headend. All nodes were static, with no mobility. The structure had the AMI headend as the gateway of the network.

Table 5.2 Smart Home-Smart Grid network protocols

<i>TCP/IP Layer</i>	<i>Smart Home Protocols</i>	<i>Smart Grid Industrial/ Commercial/ Reliability Services</i>	<i>Smart Grid AMI Services</i>
Physical	802.11	802.11, 802.30	802.30
Data-Link	SSID, Ad-Hoc	SSID, Ad-Hoc	SSID
Network	IPv4	IPv4	IPv4
Transport	TCP/UDP	TCP/UDP	TCP
Application	HTTP, MQTT, CoAP, XMPP, DLMS/COSEM	DLMS/COSEM	HTTP, MQTT, CoAP, XMPP, DLMS/COSEM

A summary of the protocols used across the TCP/IP layers is in Table 5.2 and a graphical representation is in Figure 5.1. More information regarding the development and usage of the application layer protocols is given below.

5.4.4 Smart Home-Smart Grid network legitimate traffic generation

Client-Server Architecture Coding and Design

As explained in this chapter, applications are responsible for generating network traffic between nodes. A node can be a client, a server or both. There are helper classes that assist in the creation of assigning client and server capabilities to nodes easily without having to code everything from scratch using traditional socket programming. However, in order to have a better control of how the nodes behave in generating traffic, we chose to code the socket programming components from scratch. To achieve that, we had to create our own version of application derived from the Application class.

In our smart home-smart grid network simulation, to realistically simulate the traffic, we had to introduce some form of randomness in the overall traffic generation on both the "client" and the "server" side. The random distribution that was used in the simulation was the Poisson distribution.

In NS-3, random variables using the Poisson distribution can be generated using the `ns3::ExponentialRandomVariable` Class. Randomness was introduced through Poisson

distribution in the following metrics, at the time a packet was created and sent from the client to the server, at the size of the packet generated from both the client and the server. Analysing the smart home datasets from [206] and [4], the packet length sizes ranged from 20-2559 bytes. Therefore, we constructed our own packet generator in the smart home-smart grid to generate packets ranging on the same interval using the Poisson distribution [43], [163].

Different speeds were also applied in IoT nodes to service application layer requests as well as and the time the server needed to respond.

Our simulation was constructed to represent realistic smart home traffic as much as possible. This was achieved by using the smart home traffic dataset generated by [206]. In [206] they deployed 30 different smart home devices, including smart cameras, blood pressure meters, NEST protect smoke alarm, smart sleep sensors, smart bulb, android/iPhone devices, laptops and so on. The maximum amount of packets generated per second was 58.7. Our simulation which consisted of 10 nodes generated an average of 18.5-22.1 packets per second.

Regarding the smart grid standards and guidelines, to simulate this part as realistically as possible we used the standard messages/signal times as defined by [204]. Smart meters, whether they are located in smart homes or in other domains on the smart grid report smart meter pricing the typical interval is between 15 minutes to 1 hour. Demand/response and load shedding signals could be reported as frequently every 2 seconds. Therefore, we varied a the transmission of such signals from 8 seconds to 15 minutes.

5.4.5 DDoS Simulation

In NS-3, there are no relevant libraries that have to do with malicious network behaviour, more specifically with DDoS attacks. Therefore, the behaviour of application layer flooding and slow-rate DDoS attacks had to be simulated. In this section we describe how flooding and slow-rate DDoS attacks are conducted in the application layer and how, based on this knowledge, we simulated these attacks in the smart home-smart grid network testbed in NS-3. A summary of the parameters and their perspective used for these attacks is presented in Table 5.3.

Flooding Simulation

In application layer flooding attacks, the attacker firstly successfully completes a TCP connection with the target. Then, they send a burst amount of application layer requests to the target, that are legitimate though. This results in the target being unable to respond to requests from legitimate users. There are tools such as LOIC/HOIC, HULK and GoldenEye

Table 5.3 NS-3 DDoS attacks traffic generation parameters

<i>Attack Type</i>	<i>Attackers Number/ Applications number.</i>	<i>Avg. packets/sec. before attack</i>	<i>Avg. packets/sec. during attack</i>
DDoS flooding	5 /10	33.1	90.1
DDoS flooding	10 /10	33.1	149.8
DDoS slow-rate	10/ 10	33.1	71.3
DDoS slow-rate	10/ 20	33.1	100.9

[25] that can conduct application layer flooding attacks using the HTTP protocol. Some of them send a constant rate of the same type of requests with the same size. Some though, vary various parameters such as the time of sending the request, the type of request, the size of the overall packet and other types of header fields. Therefore it is fact that it is harder to detect a DDoS attack that has varied parameters than a constant version of it.

As a result, we aimed to simulate a varied behaviour in the application layer flooding DDoS attack. The rate at which the packets are generated and sent though is exponentially increased in a short amount of time. In detail, the simulation included the client-attacker successfully completing a TCP connection. Then, the attacker sent a large amount of requests to the target. In order for the DDoS to demonstrate a more complex behaviour, we created a variation in the following parameters using the Poisson distribution: packet size, and the time the packet is sent.

Slow-rate Simulation

The behaviour of the application layer slow-rate is less straightforward compared to the flooding DDoS attack. The rate at which the packets are generated and sent through is much slower and the request is never completed.

The attacker once again successfully completes a TCP connection with the target. Then, the attacker initiates a large number of requests that are never completed though. This can normally occur in cases where the connection is slow and the request cannot be completed in one packet. As a result, the request is completed in multiple packets. Then, the server reassembles the packet and sends the appropriate response. The server can wait for a pre-defined amount of time for the client to send a new packet, before it closes the connection. The attacker exploits this vulnerability, by sending a subsequent packet so the connection can remain open. As time is passing by, more connections are initiated with incomplete requests. As a result, once again the target is unable to respond to requests from legitimate users.

As a result, we aimed to simulate a varied behaviour in the application layer flooding DDoS attack. In detail, the simulation included the client-attacker successfully completing a

TCP connection. Then, the attacker starts initiating a number of requests to the server but those shall be marked as incomplete. As a result, the server node is obliged to wait for the client to send the "final" packet for the request to be completed. Once again we varied the time of sending the request packet and packet size parameters using Poisson distribution to make it harder to detect the DDoS attack conducted.

Poisson distribution was introduced to create randomness on the request/packet transmission and packet size. The literature in network simulation has Poisson distribution as a preferred method for simulated network traffic generation so we decided to follow the same approach [43], [163].

Furthermore, as the authors in [92] effectively stated, application layer DDoS attacks are stealthier because of their intelligent execution and a requirement to be similar to legitimate connections. In that way, they can possibly avoid detection or delay their detection by security countermeasures. In practice, popular tools conducting application layer flooding and slow-rate attacks such as the ones used in [105] use randomisation techniques.

Specifically for the flooding attacks, HULK [21], DDoSim [2] and Goldeneye [189] tools were used. HULK randomises the payloads' content and size, the headers used and the type of request. DDoSim follows a simplistic approach with identical requests generated. Goldeneye can also send randomised traffic with different request types, number of concurrent connections and user-agents.

The slow-rate attacks, can be divided on focusing on either the headers or the body of the request. Slow header rate attacks can be conducted using Slowloris [253]. Slowloris can randomise the incomplete requests (either GET or POST) on their type and time of sending the request. Another popular slow header rate tool, Slowhttptest [196] provides a more general implementation that allows of multiple methods to conducting the attack and therefore providing more randomisation options. Hence, after researching the material summarised above, we used Poisson distribution to randomise the packet size and the request initiation time and number.

Increasing DDoS Attacks Simulation

Most of the DDoS attacks, flooding in particular, can be classified as constant rate. In constant rate attacks, the attackers generate a high steady rate of traffic towards the target. The impact of such an attack is fast, but it can easily be detected due to its obvious intensity. Hence, attackers moved towards more sophisticated ways of conducting DDoS attacks.

One way to evade any security measures installed, is to slowly but steadily flood the target in an increasing-rate attack, where the maximum impact of the attack is reached gradually over the attack period. Furthermore, slow-rate application layer DDoS attacks follow a

different approach in generating requests to the target. They initiate a series of incomplete requests at random times. Between the two request opening events, no new requests are opened at all. Only the initiated requests send small packets for the requests to remain open.

Unfortunately, there are no real-world DDoS attacks insights for us to get more information and use it to simulate a DDoS attack. Instead, we researched DDoS attacks present in constructed datasets from other universities and used similar rates regarding the rate of packets and packet size.

With regards to “worst DDoS scenarios”, the way we chose to simulate the DDoS attacks was after researching different variations of conducting DDoS attacks such as constant and increasing DDoS attacks presented in [159], [150]. Hence, we decided to simulate similar scenarios for our DDoS simulation for both flooding and slow-rate cases.

Furthermore, regarding the window size, we have performed an extended research on what values do window sizes take from different studies with some examples including [176], [148], [147], [71], [218]. Based on the datasets constructed of application layer DDoS attacks (or other DDoS attacks in lower layers), the shortest duration of a DDoS attack was approximately 13 minutes and the longest was approximately for 60 minutes. Therefore, we decided not to simulate a DDoS attack that is shorter than 40 seconds since no DDoS attacks of that duration were present in the datasets researched. In addition, we believe that a DDoS attack that is shorter than 40 seconds is unlikely going to cause any issues to the network due to its very short duration. For the reasons mentioned, we did not simulate a DDoS attack that is shorter than 40 seconds.

To simulate increased-rate attacks in NS-3, we initiated random connections after random time-intervals up until the half of the total duration of the attack. Randomness was also introduced to the application layer DDoS attacks behaviour through Poisson distribution. Randomness was used in the following metrics, at the time a packet was created and sent from the client to the server, at the size of the packet generated from both the client and the server and at the time the server needed to respond. Different speeds were also applied in IoT nodes to service application layer requests.

DDoS Attacks Simulation compared to real-world DDoS attacks

To simulate the DDoS attacks as realistically as possible we studied other DDoS attacks from other datasets, specifically the flooding attack from the ISCXIDS 2012 dataset and the slow-rate attack from the CICIDS 2017 dataset.

In the relevant flooding DDoS attack file of the ISCXIDS 2012 dataset, the average packet transmission before the flooding attack took place was approximately 126.8 packets/sec. by 21 legitimate users. During the flooding DDoS attack 7 of them were compromised on a

botnet and start behaving maliciously by conducting an application layer flooding DDoS attack thus increasing the traffic to 583.1 packets/sec., reaching a maximum of approximately 2550 packets.

In our case, the DDoS flooding was external and conducted by an additional 5 or 10 attackers. The average packet transmission was approximately 33.1 packets/sec. before the attack and increased to 90.1 packets/sec. and 149.8 packets/sec. respectively. Unarguably, a flooding DDoS attack conducted with IoT devices and smart appliances can by no means have the magnitude of an flooding DDoS attack on traditional ICT networks but both attacks simulated on the smart home-smart grid network have approximately the same analogy. As a result, we are confident that our simulation had similar analogy.

In the relevant slow-rate DDoS attack file of the CICIDS 2017 dataset, the average packet transmission before the slow-rate attack took place was 133.7 packets/sec. by 25 legitimate users. During the slow-rate DoS attack, 1 of them was compromised and started behaving maliciously by conducting an application layer slow-rate DoS attack thus increasing the traffic to 283.8 packets/sec.

In our case, the DDoS slow-rate was external and conducted by an additional 10 or 20 applications encompassed in 10 attackers. The average packet transmission was approximately 33.1 packets/sec. before the attack and increased to 71.3 packets/sec. and 100.9 packets/sec. respectively. Unarguably, a slow-rate DDoS attack conducted with IoT devices and smart appliances can by no means the magnitude of an slow-rate DDoS attack on traditional ICT networks. The limitation in this case was that the real slow-rate attack was a DoS and not a DDoS attack. However, by checking on the analogies we are confident that our simulation was realistic enough.

5.5 Weka Toolbox and Weka Libraries

Since ForChaosLR multi-layer IDS contains a logistic regression model as part of its second-layer implementation and since it is going to be compared with a set of different ML algorithms it was important to implement all of them with the same library to avoid any cross-implementation issues. It was also important for this library to be robust, flexible, with as few bugs as possible and available in Java since the ForChaos first-layer was implemented in Java.

One of the most popular ML tools and code libraries available in Java is Waikato Environment for Knowledge Analysis (Weka) [144]. It was developed at the University of Waikato, New Zealand and it also contains both a graphical version along with the relevant Java libraries.

Weka accepts a specific type of input, the .arff file. ARFF supports numeric and nominal (categorical) attributes. Although it does not directly provide a boolean type option for instances, nominal can be easily converted into boolean. Every .arff file starts with the symbol "@" plus the title of the dataset, indicating what kind of data are present in the file. In the example below, the title is weather because the dataset consists of weather data. Then, the features-attributes are specified using "@" with the keyword attribute. Each feature-attribute is given a name followed by its type. The type can be nominal, numeric or real. If it is nominal, the values the feature-attribute is allowed to have must be specified in curly brackets. The last attribute in the .arff file denotes the class of the dataset, which Weka is going to consider as the label for each instance. Next, to denote the start of the actual data "@" is used along with the keyword "data". Finally, the actual instances of data are defined. Normally, all instances must have a value for all the attributes specified as the majority of machine learning algorithms cannot handle missing values for attributes. However, some machine learning algorithms can handle missing values for attributes. In such cases, a "?" must be specified for each missing value. In .arff files any comments must start with "%". In Figure 5.2 we present a part of our own NS-3 binary dataset.

```
@relation Attacks
attribute 'RequestsNo' real
@attribute 'AverageTimeBetwGetGet' real
@attribute 'AverageTimeBetwOkGet' real
@attribute 'AverageTimeBetwOkOk' real
@attribute 'PacketsNumber' real
@attribute 'PacketSize' real
@attribute 'AverageDataRate' real
@attribute 'ParallelRequests' real
@attribute 'Class' {'attack', 'normal'}
@data

622.0,0.00727,0.003614,0.0078366,1185.0,175.215189,0.027684,586.0,'attack'
456.0,0.00690,0.003558,0.0074156,2302.0,142.24066,0.0436584,590.0,'attack'
428.0,0.00655,0.003886,0.0070441,2229.0,143.39928,0.0426182,590.0,'attack'
484.0,0.00953,0.005714,0.01011718,602.0,250.790697,0.0201301,369.0,'normal'
463.0,0.00892,0.005316,0.0094874,589.0,257.2173174,0.020200,387.0,'normal'
427.0,0.00835,0.004794,0.0088947,541.0,255.0073937,0.018394,397.0,'normal'
456.0,0.00789,0.005018,0.0083300,574.0,254.1445993,0.019450,405.0,'normal'
```

Fig. 5.2 Arff file source code

As stated, Weka also offers libraries written in Java for directly using them to implement various machine learning algorithms. We used the Weka libraries to implement LR second

layer as well numerous ML algorithms, which are presented in detail in Chapter 6, Table 6.19.

5.6 Summary

In this chapter we described the experimental environments and tools we used to create the experimental environment of smart home-smart grid network as well as the dataset created to evaluate our ForChaosLR multi-layer IDS. All the relevant basic concepts and knowledge regarding NS-3 and Weka were described.

Proceeding, all the relevant work in terms of coding and data generation was followed introducing the correct behaviour and parameters to the smart home-smart grid simulated environment. Due to lack of relevant libraries in NS-3, especially regarding application IoT protocols, we had to create our own libraries to use and all the relevant and available standards and guidelines were considered and followed regarding transmission rates. Furthermore, real-world DDoS attacks were studied so we could simulate the DDoS attacks on the smart home-smart grid environment as realistically as possible.

In addition, through the smart home-smart grid simulated environment in NS-3 we generated a large set of legitimate and malicious data so we could create a suitable dataset. This dataset will be used to train and evaluate ForchaosLR and the ML algorithms. We processed the dataset so it could be used by Weka as an acceptable input. Next, we used the dataset to train the ML algorithms and LR second-layer. We also used the relevant Weka libraries to implement the LR second-layer as well as all of the ML algorithms needed.

Chapter 6

ForChaosLR Evaluation using the NS-3 Binary Dataset

6.1 Overview

In this chapter, we include the full procedure of training and testing ForChaosLR multi-layer IDS using the NS-3 binary dataset. More details regarding the dataset can be found in Section 5.2.1. The main objective of this chapter was to initially train both ForChaos and LR layers to obtain the optimal parameters for both of them and then evaluate the ForChaosLR multi-layer IDS as a whole system using the simulated smart home-smart grid network traffic.

During training, for ForChaos first-layer we performed an extended series of experiments to decide on the optimal values for the window size w and on the α value. For LR second-layer we performed an extended series of experiments to decide on the optimal values on the logistic regression function coefficients and constant values.

During testing, we evaluated ForChaosLR multi-layer IDS through a plethora of different scenarios. These scenarios were simulated in our smart home-smart grid network simulation in NS-3. These include both flooding and slow-rate attacks as well as combinations of them. Furthermore, multiple parameters regarding these scenarios were frequently changed in an attempt to assess ForChaosLR multi-layer IDS's accuracy; these parameters include number of attackers/applications, rate of attack, attack start time and attack duration. In addition, flash crowd event scenarios were used to evaluate ForChaosLR multi-layer IDS's response against flash crowd events.

To assess ForChaosLR multi-layer IDS's accuracy we used multiple evaluation metrics including as true positive rate (TP rate), false positive rate (FP rate), precision and F-score. We also compared ForChaosLR multi-layer IDS against a variety of different machine learning

(ML) algorithms. Proceeding, we constructed the list below, summarising the sections in this Chapter.

Evaluation metrics: In this section we present the evaluation metrics used to evaluate ForChaosLR multi-layer IDS's and ML algorithms' accuracy.

Scenarios independent variables: In this section we describe the independent variables manipulated in the experiments presented in this chapter.

ForChaosLR multi-layer parameters to be optimised in training: In this section we describe the relevant parameters of ForChaosLR multi-layer IDS that are to be optimised during training.

Training of ForChaos first-layer: In this section we present and describe in detail the training procedure that was followed to optimally train ForChaos first-layer.

ForChaos first-layer testing and evaluation: In this section we present and describe in detail the testing procedure that was followed to effectively and optimally evaluate ForChaos first-layer. The scenarios include attack scenarios, increasing attack scenarios, mixed attack scenarios, restart attack scenarios and flash crowd events.

Training of LR Second-Layer: In this section we present and describe in detail the training procedure that was followed to optimally train LR second-layer.

LR second-layer testing and evaluation: In this section we present and describe in detail the testing procedure that was followed to effectively and optimally evaluate LR second-layer.

Summary of ForChaosLR against NS-3 binary dataset: In this section, we summarise the overall results of ForChaosLR multi-layer IDS against the NS-3 dataset.

Evaluation of ForChaosLR against popular machine learning algorithms: To evaluate the accuracy of ForChaosLR multi-layer IDS in further, we used other popular machine learning to detect application layer flooding and slow-rate attacks. Specifically, the algorithms used were the following: Bayesian networks (BN), naive Bayesian (NB), K-nearest neighbour (KNN), support vector machine (SVM), decision tree (DT), random forest (RF) and artificial neural networks with the multi-layer perceptron architecture (MLP).

Summary: Finally, we summarise the overall results of the chapter before proceeding to Chapter 7.

6.2 Evaluation Metrics

In the intrusion detection literature as presented in Chapter 3, studies use an extended series of evaluation metrics and statistical tests to assess their detection engines performance. Measuring the system's detection rate is not sufficient, therefore different metrics are adopted to provide a more holistic evaluation. In this section, we briefly describe the most popular

evaluation metrics, the actual formulas and their application in intrusion detection. In intrusion detection research and literature, the presence of a cyberattack is regarded as a positive instance and the absence of it as a negative instance. For the correct assessment of results multiple statistical measures were adopted. Below, we mention and briefly describe the most relevant ones [14].

True Positive Rate: Otherwise called recall or sensitivity presented in Equation 6.1, is when the detection algorithm is able to correctly identify an instance as an attack.

$$TPR = \frac{TP}{TP + FN} \quad (6.1)$$

False Positive Rate: Otherwise called fall-out presented in Equation 6.2, is when the detection algorithm incorrectly classified a normal instance as an attack.

$$FPR = \frac{FP}{FP + TN} \quad (6.2)$$

Precision: Otherwise called positive predictive value presented in Equation 6.3, denotes the rate at which the detection algorithm correctly classified true positive instances out of all the positive instances it classified. In general, precision denotes the fraction of instances that are relevant from the total number of all retrieved instances.

$$Precision = \frac{TP}{TP + FP} \quad (6.3)$$

F1-Score: Otherwise called F1-measure or harmonic mean presented in Equation 6.4, measures an algorithm's trade-off between recall and precision. It considers true positives, false positives and false negatives to compute the score. An F1-score can take values from 0 to 1. However, an F-score can be misleading as it does not consider the true negatives at all.

$$F1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (6.4)$$

6.3 Scenarios Independent Variables

In order for us to evaluate the accuracy of ForChaosLR multi-layer IDS and the ML algorithms we have to evaluate it through a series of different sets of evaluation experiments. One of the most important factors to consider in experiments, is the manipulation of independent variables. In an experiment, independent variables denote the factors that when changed they directly have an impact on the dependent variable, in our case our proposed IDS's accuracy. Below, we present and briefly describe these independent variables that were manipulated

in the scenarios presented in this chapter. A summary of the independent variables used in the scenarios along with their respective values is presented in Table 6.1 and summarised in Figure 6.1.

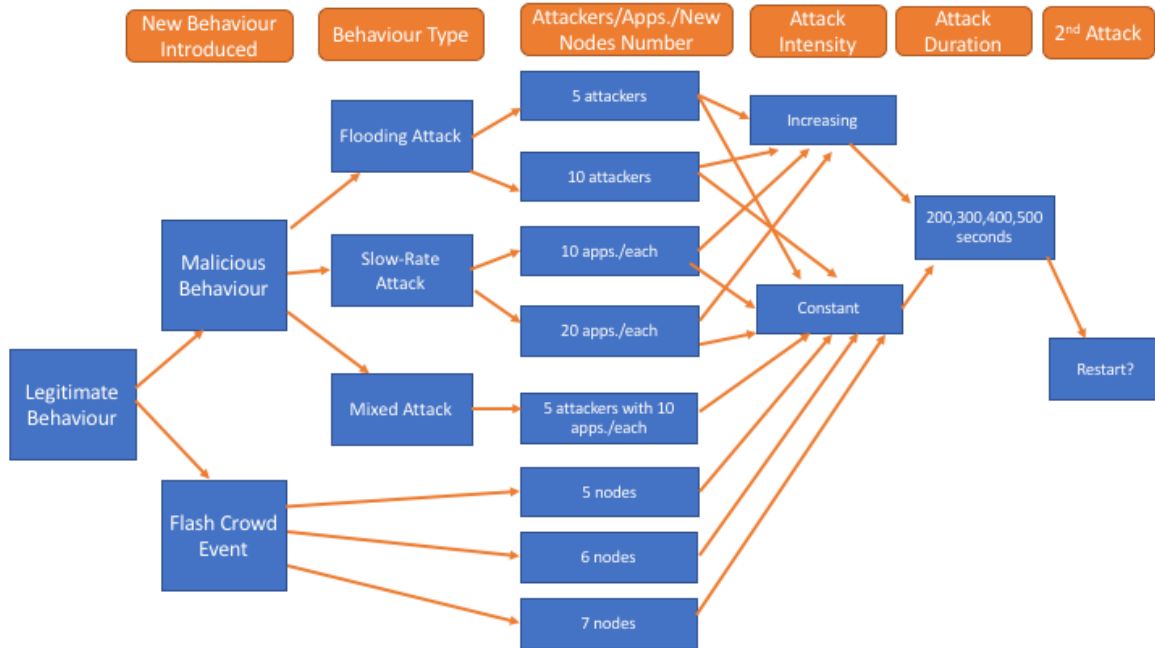


Fig. 6.1 Experiments Independent Variables Diagram for NS-3 smart home-smart grid dataset

Table 6.1 NS-3 smart home smart grid Scenarios Independent Variables

Variable	Values
Window Size w	30,40,60 (seconds)
Constant α	0.1-0.9
Behaviour Type	Flooding, Slow-Rate, Mixed, Flash Crowd
Attack Intensity	Constant, Increasing
Attack Duration	200, 300, 400, 500
Attackers Number	5,10
New Nodes Number	5, 6, 7
Applications Number	10, 20
Second Attack Performed	true/false

6.3.1 Window Size w

Window Size is a fundamental variable in both ForChaos and LR layers of our IDS. It denotes how frequently the IDS collects network traffic so the relevant features can be extracted

and analysed for possible DDoS attack. Similar studies evaluating a window size on their forecasting algorithm to detect DDoS attacks studies used an empirical evaluation to choose the most suitable window size [176], [148], [147], [71], [218]. The maximum amount of window size observed in related studies is 5 minutes. Therefore, we aimed to evaluate different window sizes up to 300 seconds.

6.3.2 Constant α

Constant α directly affects the performance of ForChaos first-layer, specifically the forecasting part of the algorithm. It is important for us to find the most optimal values that will manage to detect both flooding and slow-rate attacks regardless of their intensity and impact but also minimise the possible FP rate.

The possibility of an adaptive alpha instead of a constant alpha was considered. However, after researching more in the literature, we concluded that an adaptive alpha is a more suitable choice for predicting network traffic but may prove an obstacle for detecting malicious traffic such as a DDoS attack [220], [156].

An adaptive α aims to improve the overall network traffic predictions produced so they can produce a value that has as little numerical difference as possible from the actual network traffic. However, in the event of a DDoS attack occurring, an adaptive alpha value can mistakenly cause the forecasting algorithm to change its prediction model so it can “improve” its prediction values which is malicious. As a result, the prediction model might initially detect a DDoS attack but then, with the adaptive alpha, change its prediction values to reflect to the traffic observed and miss the rest of the attack.

A positive Lyapunov exponent categorises incoming traffic as malicious or not in ForChaos first layer. The Lyapunov exponent measures the separation between trajectories, in our case the two trajectories are the prediction for the network traffic and the actual network traffic values. The rate of separation between them is the error generated from their subtraction (in practice it is the distance between them). Therefore, the greater the distance, the higher the chance for the Lyapunov exponent to be positive. A positive alpha assists in the prediction value to be as closely as possible to the incoming traffic value. This can help when the traffic is attack-free and produce a lower FP rate. However, an adaptive alpha is likely to make things difficult in detecting DDoS attacks (especially when their intensity is initially low to avoid detection) since the error is going to be substantially reduced and therefore, the distance between the two trajectories will be reduced. This is likely to cause the Lyapunov exponent to falsely remain negative and unable to detect the DDoS attack.

6.3.3 Behaviour Type

All scenarios presented in this chapter (from the NS-3 dataset), consisted of legitimate network behaviour followed by new behaviour which was either malicious (flooding, slow-rate, mixed) or legitimate but different to previous behaviour observed (flash crowd). Legitimate behaviour in the system does not have a distinct behaviour, In contrast, it is a spectrum of different types of behaviour. Thus, due to the versatility of normal behaviour and also the nature of the application layer DDoS attacks, it is important to evaluate the system's accuracy in distinguishing between normal and malicious behaviour.

Flooding: In these type of scenarios all the attackers present in the network have conducted an application layer flooding DDoS attack.

Slow-Rate: In these type of scenarios all the attackers present in the network have conducted an application layer slow-rate DDoS attack.

Mixed: In these type of scenarios all the attackers present in the network have conducted a mixture of application layer flooding DDoS attack and slow-rate DDoS attack.

Flash Crowd: Application layer flooding DDoS attacks are very similar to flash crowd traffic. Flash crowd denotes the entrance of sudden burst of traffic in the network that is legitimate. It is hard to distinguish flash crowd events from application layer DDoS attacks because both of them generate a high amount of requests in a relatively short amount of time. We aimed to evaluate ForChaos algorithm through a series of flash crowd scenarios. To simulate random flash crowd event behaviour we used the Poisson distribution. However, the rate of sending packet was increased from the normal behaviour, but not being equal or exceeding the metrics of the flooding attack.

Based on study by [138] we proceed by giving our rationale on why we chose to conduct the flash crowd behaviour in that way below. According to [138] one of the main differences between DDoS and flash crowd is the following: in flash crowd events the requests to the server from different users (or devices in our case) is increased gradually and then it peaks. However, in the event of a DDoS attack all attackers/zombies are likely to launch a large number of requests to the target simultaneously or within a very short amount of time. In IoT networks, the number of devices that can be connected from inside is unlikely to be the same number of devices from outside that are used in a DDoS attacks. In a DDoS attack, the number of zombies is likely to be considerably high while in a smart home network, the number of new devices to be added is highly unlikely to be of that high quantity. We believe that use-cases of sudden change in traffic can occur, resulting from everyday human-based scenarios happening inside the smart home. Examples include: the installation of a set of new additional smart home appliances (e.g. new sets of smart kitchen appliances, new set of lighting installations, outdoors/garden smart sensors etc.) or the arrival of guests bringing

their own devices to connect to the network (e.g. group online gaming, social gatherings etc.).

Based on this justification by this study as well as our rational, we chose to use the same rate of requests sent as the one used in legitimate behaviour that is not flash crowd. Of course we should acknowledge that the adversary could choose to confuse the IDS by gradually increasing the requests sent as the authors in [159] and [150] effectively considered and evaluated. This approach is more sophisticated and less likely to be detected, at least immediately. Hence, we also constructed the DDoS increasing attack scenarios to evaluate on whether our proposed system could identify the difference.

6.3.4 Attack Intensity

Most of the DDoS attacks, flooding types in particular, can be classified as constant rate. In constant rate attacks the attackers generate a high steady rate of traffic towards the target. The impact of such an attack is fast, but it can easily be detected due to its obvious intensity. Hence, attackers moved towards more sophisticated ways of conducting DDoS attacks. One way to evade any security measures installed is to slowly but steadily flood the target in an increasing-rate attack, where the maximum impact of the attack is reached gradually over the attack period [159], [150].

Constant-rate attack: The maximum number of resources is used immediately when the attack is started.

Increasing-rate attack: The maximum number of resources is used gradually over the attack period. In all the scenarios, the attack is gradually achieved through half of the attack duration.

6.3.5 Attack Duration

A DDoS attack, as any other kind of attack, can be varied in its attack duration. In our experiments the attack duration in every type of experiment varied from 200 to 500 seconds with 100 seconds interval. As a result, for every experiment, there was a respective scenario with the attack lasting 200, 300, 400 and 500 seconds.

6.3.6 Number of Attackers

In the experiments which flooding malicious behaviour was used, the number of attackers was either 5 or 10 in all of the scenarios conducted.

6.3.7 Number of New Nodes

In the experiments which flash crowd behaviour was used, the number of new nodes was either 5, 6 or 7 in all of the scenarios conducted.

6.3.8 Number of Applications

In NS-3, the "apps" concept is roughly mapped to the real-world web session threads and since we are focusing at the application layer, it is sensible to change the number of applications. The number of applications used in all of the experiments was either 10 or 20 per attacker.

This parameter is also manipulated based on the intensity parameter since it consists of the time the applications were initiated. If the parameter was set to "constant", then all of the attackers' applications were initiated at the start. If the parameter was set to "increasing", then the attackers' applications were gradually initiated. Through this differentiation, we aimed to test if ForChaosLR could detect the attack, even when the peak time of the attack was not visible.

6.3.9 Second Attack Performed

Real-world DDoS attacks incidents were reported in the past where the adversary has re-initiated a DDoS attack towards the target [174] and [145]. In general, it is important for an IDS system not to only detect malicious behaviour but get more information regarding the time conducted and the duration of it as well as multiple DDoS attacks conducted in a space of time. Since DDoS attacks that stop and then re-initiated is a plausible scenario based on the real-world cases stated, we believe it is an important scenario to evaluate our proposed solution. Furthermore, ForChaos first-layer uses simple exponential smoothing forecasting which constructs its future observations based on previous observations. If a DDoS attack occurs, it is likely that when the attack is finished the model will continue flagging up incoming traffic, even though it is legitimate due to the fact it does not look similar to previous observed behaviour (the DDoS attack observations). During this set of experiments, we wanted to investigate what variables can affect ForChaos first-layer's behaviour in falsely classifying incoming legitimate traffic as malicious (attack start time, duration and magnitude) as well as if this behaviour can negatively impact ForChaos first-layer effectiveness towards a second DDoS attack occurring in a short amount of time. As a result, we simulated scenarios where an attack, regardless of the type and/or intensity, was conducted and then the network is returned to its normal state. The attack duration was 200, 300, 400 or 500 in the scenarios simulated.

Table 6.2 ForChaosLR multi-layer IDS Training Parameters for NS-3 smart home-smart grid dataset

<i>Variable Name</i>	<i>ForChaosLR/ Layer</i>
Window Size w	First Layer (ForChaos)
Constant α	First Layer (ForChaos)
Training Time	First Layer (ForChaos)
RequestsNo Coefficient	Second Layer (LR)
Avg TimeBetwGetGet Coefficient	Second Layer (LR)
Avg TimeBetwOkGet Coefficient	Second Layer (LR)
AverageTimeBetwOkOk Coefficient	Second Layer (LR)
PacketsNo Coefficient	Second Layer (LR)
Avg PacketSize Coefficient	Second Layer (LR)
AverageDataRate Coefficient	Second Layer (LR)
ParallelRequests Coefficient	Second Layer (LR)
Constant Coefficient	Second Layer (LR)
Ridge Value	Second Layer (LR)

6.4 ForChaosLR multi-layer Parameters to be Optimised in Training

Training is very important step in the construction of ForChaosLR multi-layer IDS for producing the best results possible during the evaluation stage. ForChaosLR must be robust, accurate and precise. To achieve this, through training we must obtain the optimal values for the variables present in our algorithmic design presented in Chapter 4. A summary of these variables in presented in Table 6.2 along with the layer that they belong to. Furthermore, due to the algorithmic nature of the two layers, different training procedures need to be followed for each one. The parameters α and w are described in detail in Chapter 4, ForChaos first-layer training and the rest of the parameters throughout Chapter 4.

6.5 Training of ForChaos first-layer

6.5.1 Overview

ForChaos first-layer is trained with only real-time data for a period of time so it can gather enough data for its forecasting model to start making accurate predictions. During this period of training, optimal values for the relevant variables used are decided, specifically α , the

window size w and training time. The parameters α and w are described in detail in Chapter 4.

As presented in Chapter 4, we chose the forecasting algorithm for ForChaos first-layer to be simple exponential smoothing forecasting based on experiments with various forecasting algorithms. Various forecasting techniques were proposed and used in the past for the effective detection of DDoS attacks with the most popular being:

1. Moving Average (MA)
2. Weighted Moving Average (WMA)
3. Simple Exponential Smoothing Forecasting (also Exponential Weighted Moving Average (EWMA))
4. Double Exponential Smoothing Forecasting (DEF)
5. Triple Exponential Smoothing Forecasting (TESF) (also Holt's Winter Smoothing).

MA and WMA make a forecast solely based on previous observations only, while the rest of the techniques consider both past observations and forecasts. We chose to exclude MA and WMA because there should be a balance between past forecasting and observations for a forecasting model to make effective and accurate forecasts. Our argument is supported by [71] which makes an evaluation study between MA, WMA, and simple exponential smoothing forecasting.

It concludes that simple exponential smoothing forecasting is the most effective forecasting algorithm as opposed to the other two against DDoS attacks. Between the three exponential smoothing forecasting models there is one basic difference between each other that assists us into making our final decision. Simple exponential smoothing forecasting has no way of considering trend if there is present in observations while double exponential smoothing forecasting considers the potential trend in the formula. Triple exponential smoothing forecasting also considers trend in data and seasonality. Although these additional elements are useful in other applications(e.g. making stock market predictions) there is no need to adopt seasonality in our network traffic prediction model.

However, trend might potentially positively affect the detection of DDoS attacks. Hence, we conducted a series of experiments using both double exponential smoothing forecasting and simple exponential smoothing forecasting to investigate which of the two is more effective in detecting application layer flooding and slow-rate DDoS attacks. Simple exponential smoothing forecasting achieved better results than double exponential smoothing forecasting so we continue our experiments using only the former forecasting method for ForChaos

first-layer. The experimental results from this set of experiments are omitted since they do not substantially contribute to the thesis' results presented or the ForChaosLR multi-layer IDS proposed system.

The three variables that need to be optimised are: window size w , constant α and training time. For the training procedure we divided the training into two phases: one for finding the optimal values for a and w and one for finding the optimal training time. The training procedures as well as their results are presented in this section. In the experiments which the results consisted of a 100% TP rate and a 0% FP rate the relevant results were omitted since these were the desirable results.

6.5.2 Evaluation of a and window size w

Motivation

As presented in Chapter 4, for ForChaos first-layer two of the variables we need to optimise are the following: α , window size w . Finding an optimal training, α and window size w is going to help the model make predictions about the traffic state as accurately as possible.

Experimental Set Up

For this training procedure we generated a number of scenarios containing legitimate and DDoS traffic to optimise α , window size w and training time needed. The specific scenarios with their respective results for each window size are in Table 6.3. We chose to evaluate window size w with the following values: 30, 40 and 60 seconds simulation time and constant α with the following values: 0.1 - 0.9 (with 0.1 interval).

The first and most important step of the training procedure was to ensure that ForChaos can detect both flooding and slow-rate DDoS attacks. Initially, we aimed to test which combinations of window sizes and w and constant α were able to detect constant DDoS attacks which was the most common and easy to detect. We set the attack duration to 200 seconds and a constant type of DDoS for different attack start times, specifically at the 1000th, 2000th, 3000th and 4000th second. Regarding training time, we evaluated the most optimal training time which produced the best combination of w , α and RMSE (root square mean error) and how they behave with regards to the actual forecasts and their respective values.

Results Analysis and Discussion

Based on the results obtained, we list our main observations and conclusions below.

Table 6.3 ForChaos first-layer Training Scenarios for NS-3 smart home-smart grid dataset

<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Rate</i>	<i>Attack Start Time (Seconds)</i>
Flooding	5/10 attackers	Constant	1000
Flooding	5/10 attackers	Constant	2000
Flooding	5/10 attackers	Constant	3000
Flooding	5/10 attackers	Constant	4000
Slow-Rate	10 attackers & 10/20 apps. /each	Constant	1000
Slow-Rate	10 attackers & 10/20 apps./each	Constant	2000
Slow-Rate	10 attackers & 10/20 apps. /each	Constant	3000
Slow-Rate	10 attackers & 10/20 apps./each	Constant	4000
Mixed	5 attackers & 10 apps/each	Constant	1000
Mixed	5 attackers & 10 apps/each	Constant	1000
Mixed	5 attackers & 10 apps/each	Constant	2000
Mixed	5 attackers & 10 apps/each	Constant	3000
Mixed	5 attackers & 10 apps/each	Constant	4000

For the attack start time on the 1000th and 3000th second, we observe that all three window sizes with $a = 0.1$ correctly detect the attacks regardless of the attack time. We only list the results from scenarios which the TP was not 100% and the FP was not 0% for space reasons since the results do not illustrate anything else apart from the fact that these specific window sizes achieve a 100% TP rate and a 0% FP rate. Therefore, the results were omitted.

If the attack start time is set on the 2000th second, we note that the 30-second window fails to detect the attack for its entire duration. In contrast, the other two window sizes (40 and 60 seconds) are unaffected by the change on the attack start time as presented in Table 6.4 seconds attack start time and $a = 0.1$.

However, in the experiments which the attack start time on the 4000h second, we observe that the 30-second window size once again is not able to detect the attack fully as opposed to the other two window sizes. Despite this, the 30-second window size clearly achieved worse results than the other two window sizes.

For that reason, we decided to evaluate the same three window sizes using a different value for a , specifically we set a to 0.2. However, the results obtained were worse for all three window sizes so we decided to proceed for the rest of our experiments with the 40 and 60 second window sizes and with $a = 0.1$.

It is evident that attack start time plays an important role in the TP rates rate across all window sizes. This conclusion is more evident in the results presented in Tables 6.6 and Table 6.8. This is directly linked with the window size chosen and the attack start time.

To explain this relationship in further detail, we are going to explain it through a specific example.

Supposedly, we have an attack start time at the 4000th second and we aimed to evaluate the 40-second and the 60-second window sizes. When the 40-second window size is chosen, a new forecast is made every 40 seconds and when the 60-second window size is chosen, a new forecast is made every 60 seconds. Hence, when the attack starts at the 4000th second, the first forecast which will contain malicious data will be made on the 4000th second (if the 40-second window size is chosen) and on the 3960th second (if the 60-second window size is chosen).

Hence, it will be harder for ForChaos first-layer with a 60-second window size to effectively correctly classify that specific forecast (3960th) second as malicious since the majority of data considered is legitimate and only the last 20 seconds are malicious (duration of the 60-second window size: 3960 to 4020 seconds). In contrast to this example, the 40-second window size will not have this issue as it will be able to capture the malicious behaviour from its start. This is clearly a practical problem that could be present in choosing whatever window size since in a real-world scenario we will not have any indication on the exact time a DDoS attack is conducted.

We have to point out that although the 0% FP rate might seem odd, the main reasons for this result throughout this chapter occurred for the following reason: In the smart home-smart grid network the network traffic observed is more stable and more predictable than an ICT network and fewer bursts of traffic are observed.

Network traffic in ICT networks would typically consist of sudden changes in traffic due to the dynamic population of devices and hosts observed in such networks while this is less likely to occur in resource-constrained networks. Furthermore, wireless ICT networks are vastly different to wireless resource-constrained networks as stated in [3] home-smart grid network. Some examples of resource-constrained networks include their data rate having a maximum rate of 250 kbps/sec for ZigBee networks (a well known and popular wireless protocol for resource-constrained networks) and a 11,000+ kbps for WiFi networks. Therefore, the network traffic monitored in the smart home-smart grid network could potentially be stable and predictable, which makes it easier for a forecasting algorithm to produce more accurate forecasts and not achieve low FP rates.

Furthermore, related studies effectively argued and supported that IoT traffic is different from ICT traffic [84],[157], [219], [207], [206], [130], [69], [201]. One of the differences denotes that IoT networks (especially smart home IoT networks) demonstrate higher heterogeneity than ICT networks. In addition, according to [157] IoT traffic can be periodic with periodic communications and event-based, with some applications being always event-driven.

Table 6.4 ForChaos Experimental Results against Constant Attacks (2000 and 4000 seconds attack start time and $\alpha = 0.1$)

<i>Attack Type</i>	<i>Attackers Number./ Applications Number</i>	<i>Window Size (Seconds)</i>	<i>TP Rate (%)</i>	<i>FP Rate (%)</i>
Flooding	5 attackers	30	85.7	0
Flooding	10 attackers	30	100	0
Slow-Rate	10 apps.	30	85.7	0
Slow-Rate	20 apps.	30	100	0
Flooding	5 attackers	40	100	0
Flooding	10 attackers	40	100	0
Slow-Rate	10 apps.	40	100	0
Slow-Rate	20 apps.	40	100	0
Flooding	10 apps.	60	100	0
Flooding	5 attackers	60	100	0
Slow-Rate	10 attackers	60	100	0
Slow-Rate	20 app.s	60	100	0

Furthermore, as the studies in [219] and [201] effectively stated, IoT network traffic follows a stable and rather predictable network traffic pattern compared to ICT networks.

However, there is always a possibility that the traffic observed in the smart home-smart grid network experiences sudden changes in the overall network traffic state that are still legitimate such as a flash crowd event. This is likely to make a forecasting algorithm produce a considerable FP rate. To evaluate this assumption we included a series of experiments where we introduce flash crowd event behaviour in the smart home-smart grid network in the ForChaos against flash crowd experiments section.

Another case which ForChaos first-layer might achieve a considerable FP rate is on the restart attack experiments presented in the ForChaos against restart attack experiments section. There were multiple DDoS attacks conducted in multiple phases for numerous reasons such as a hacktivist agenda, ransom payment, damage to assets and so on [174], [145].

6.5.3 Evaluation of ForChaos first-layer Training Time

Motivation

Training time is a very important measure to evaluate, especially when dealing with forecasting algorithms. Since ForChaos first-layer uses simple exponential smoothing forecasting, it was essential to decide on a reasonable amount of training time. During this time, ForChaos

would observe enough data to make accurate predictions. These predictions would not only assist in detecting DDoS attacks, but also identify when a temporary network traffic abnormality is legitimate.

Experimental Set Up

For this series of experiments we focused on the actual forecasts made by the three different sizes and what error they produced, specifically using the root mean square error (RMSE). We generated three sets of legitimate traffic (total of 3000 seconds) to evaluate on how effective each window size is in predicting the future network state.

Results Analysis and Discussion

In Figures 6.2, 6.3 and 6.4 we present three graphs for the 30, 40 and 60-second window sizes of ForChaos first-layer. These three graphs illustrate how the error generated at every prediction behaves over the period of 3000 seconds. From the graphs, we observe the RMSE error starts decreasing asymptotically at approximately the 1000th to 1200th second for all three window sizes. Based on these conclusions, we decided to set the training time to 1000 seconds.

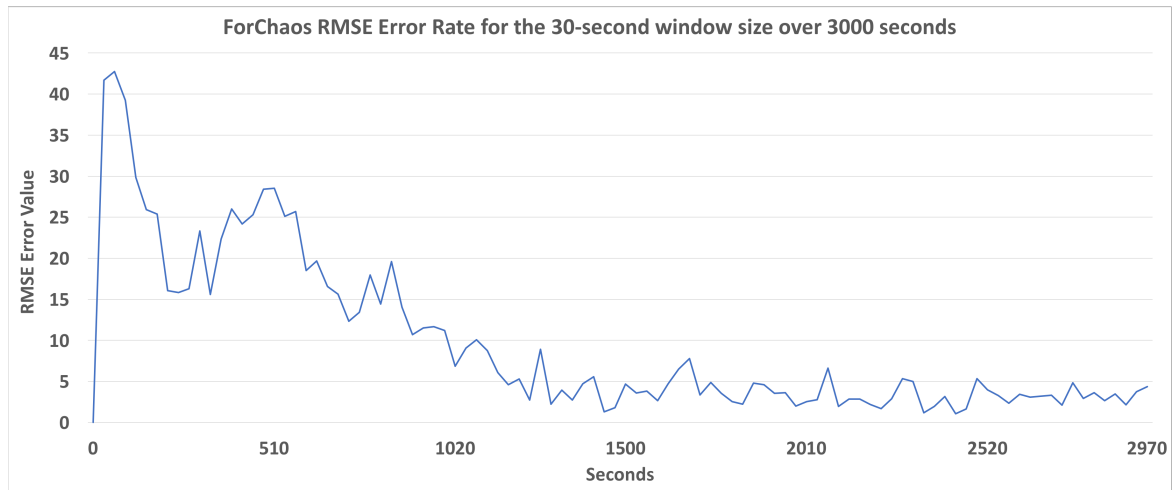


Fig. 6.2 30-second Window Size RMSE Error Rate for 3000 seconds for NS-3 smart home-smart grid dataset



Fig. 6.3 40-second Window Size RMSE Error Rate for 3000 seconds for NS-3 smart home-smart grid dataset

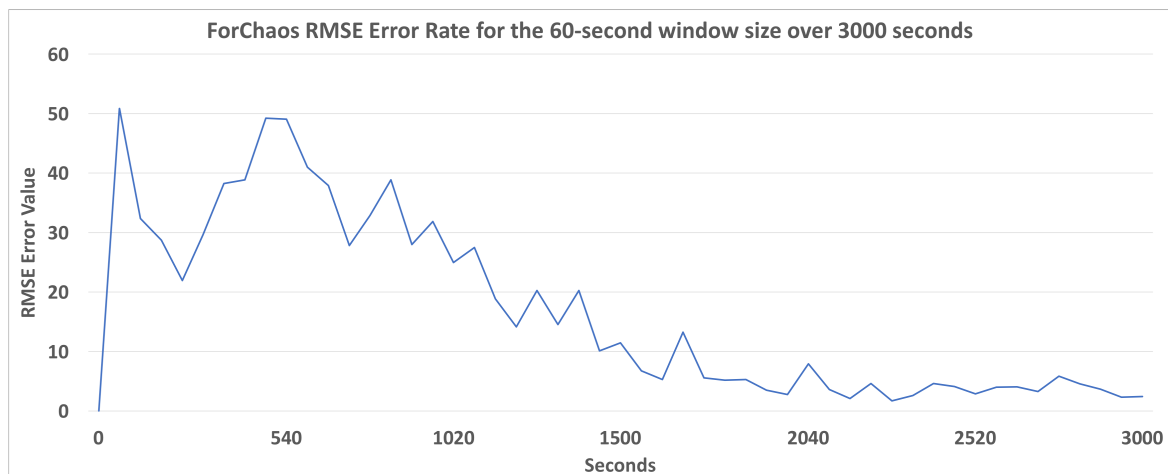


Fig. 6.4 60-second Window Size RMSE Error Rate for 3000 seconds for NS-3 smart home-smart grid dataset

6.6 ForChaos first-layer Testing and Evaluation

6.6.1 Overview

In this section, the overall results of the ForChaos first-layer detection are presented. To assess the accuracy of ForChaos we simulated a number of different experiments by modifying one or more of the independent variables presented in Table 6.1. Both the 40-second and the 60-second window sizes were evaluated so we could conclude which window-size performs best.

6.6.2 ForChaos against Constant Rate DDoS Attacks Experiments

Motivation

In the event of a DDoS attack, the set of attackers are likely to start attacking the target with all of their resources instantly and for the entire duration of the attack since their objective is to make the target unavailable as soon as possible. Based on this concept we aimed to evaluate if the ForChaos was able to detect both constant rate DDoS attacks.

Experimental Set Up

For evaluating ForChaos, we constructed multiple scenarios presented in Table 6.5. In these scenarios the following variables have been used as part of the experimental evaluation phase: duration of the attack and its start time. Regarding the duration of time, we aimed to check whether a longer attack could be detected in all of its duration or not since ForChaos might fail to detect a network traffic state after a while due to it being dependent on previous forecasts.

Regarding attack time, we aimed to check whether an attack conducted at a much later time can detect by ForChaos. ForChaos uses forecasting algorithms (along with Lyapunov exponents) to classify a network traffic state based on previous observations. The more observations it observes the more difficult it might be for it to classify a malicious traffic state as a DDoS attack, especially if the network state is prone to sudden changes on incoming traffic. We tested both the 40-second and the 60-second window sizes to evaluate their accuracy.

6.6.3 Results Analysis and Discussion

The results from the constant attack scenarios are presented in Table 6.6. Proceeding, we summarise the most important findings regarding this series of experiments.

Table 6.5 ForChaos scenarios for constant attacks (40 and 60 seconds window size with $\alpha = 0.1$)

<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>
Flooding	5/10 attackers	1000	200,300,400,500
Flooding	5/10 attackers	2000	200,300,400,500
Flooding	5/10 attackers	3000	200,300,400,500
Flooding	5/10 attackers	4000	200,300,400,500
Slow-Rate	10/20 apps. /each	1000	200,300,400,500
Slow-Rate	10/20 apps./each	2000	200,300,400,500
Slow-Rate	10/20 apps. /each	3000	200,300,400,500
Slow-Rate	10/20 apps./each	4000	200,300,400,500

Table 6.6 ForChaos first-layer results against constant attacks - 40 and 60 seconds window size and $\alpha = 0.1$ results

<i>Window Size (Seconds)</i>	<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>	<i>TP Rate (%)</i>	<i>FP Rate (%)</i>
40	Slow-Rate	10 apps./each	1000	500	83	0
40	Slow-Rate	10 apps./each	2000	400	80	0
40	Slow-Rate	10 apps./each	2000	500	60	0
40	Slow-Rate	10 apps./each	3000	400	90	0
40	Slow-Rate	10 apps./each	3000	500	75	0
40	Slow-Rate	10 apps./each	4000	400	90	0
40	Slow-Rate	10 apps./each	4000	500	75	0
60	Flooding	5 attackers	4000	200	75	0
60	Flooding	5 attackers	4000	300	80	0
60	Flooding	5 attackers	4000	400	86	0
60	Flooding	5 attackers	4000	500	89	0

- Both the 40-second and the 60-second window sizes achieved satisfactory results with a TP rate more than 60% in all the experiments and a 0% FP rate. In Table 6.6 we only list the results from scenarios which the TP was not 100% and the FP was not 0% for space reasons since the results do not illustrate anything else apart from the fact that these specific window sizes achieve a 100% TP rate and a 0% FP rate.
- Regardless of the window size, attack type, intensity, attack start time and duration, ForChaos first-layer is robust and does not generate any FP rates. Both the 40-second window and the 60-second window size achieve a high TP rate across the different attack scenarios which they were evaluated. We expected ForChaos to demonstrate problems identifying the attacks with weaker intensity, specifically flooding with only 5 attackers and slow-rate with only 10 apps used on each attacker.
- Evidently, the 40-second window size had issues identifying slow-rate attacks across all four attack start times and in particular, the 400 seconds and 500 seconds attack duration scenarios. Based on results, we understand that although the results worsen when the attack start time is at the 2000th second for both the 400 and 500 seconds attack duration, they improve for the 3000th and 4000th attack start times. This indicates that the more observations the 40-second window size has, the more accurate it is in classifying incoming traffic as malicious. The 40-second window size has no issues identifying the rest of the attacks: flooding with 5 or 10 attackers and slow-rate with 20 apps. deployed on each attacker.
- Related studies that use forecasting algorithms [225], [121], [16], acknowledge and support that forecasting need to be retrained at some point so they can continue producing accurate predictions.
- The reason is that such models are directly dependent on previous observations to make their predictions. Since network traffic contains natural, and possibly frequent, variations, its current state might be different from previous states. Hence, the forecasting algorithm needs to be retrained.
- The 60-second window size has issues identifying flooding attacks during the 4000th second attack start time for the 200, 300, 400 and 500 seconds duration attacks. Evidently, the later the attack start time is, the more negative impact it has on the 60-second window size regardless of the attack duration.

Table 6.7 ForChaos first-layer results against increasing attacks - 40 and 60 seconds window size and $\alpha = 0.1$

<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Rate</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>
Flooding	5/10 attackers	Increase	1000	200,300,400,500
Flooding	5/10 attackers	Increase	2000	200,300,400,500
Flooding	5/10 attackers	Increase	3000	200,300,400,500
Flooding	5/10 attackers	Increase	4000	200,300,400,500
Slow-Rate	10/20 apps. /each	Increase	1000	200,300,400,500
Slow-Rate	10/20 apps./each	Increase	2000	200,300,400,500
Slow-Rate	10/20 apps. /each	Increase	3000	200,300,400,500
Slow-Rate	10/20 apps./each	Increase	4000	200,300,400,500

6.6.4 ForChaos against Increasing Rate DDoS Attacks Experiments

Motivation

Due to constant attacks generating high levels of "noise", their detection is relatively easy for an IDS.

Hence, an adversary's goal is to conduct the DDoS attack and remain undetected for as long as possible. Therefore, the attack conducted can be stealthy and does not make "noise". Hence, the adversary is going to gradually conduct the attack by opening the requests to the target gradually and not all of them at once. Based on this concept we aimed to evaluate if the ForChaos was able to detect increasing rate DDoS attacks.

Experimental Set Up

For evaluating ForChaos, we once again constructed multiple scenarios presented in Table 6.7. In these scenarios the following variables were used as part of the experimentation phase: duration of the attack and attack start time.

Regarding duration of time, we aimed to check whether a longer attack in time could be detected in all of its duration or not since ForChaos might fail to classify a network traffic state as an attack at the beginning since the attack has an increased rate and initially it might seem as a legitimate change in the network traffic state.

Regarding attack time, we aimed to check whether an attack that is conducted at a much later time is able to be detected by ForChaos. ForChaos uses forecasting algorithms (along with Lyapunov exponents) to classify a network traffic state based on previous observations. The more observations it has observed the more difficult it might be for it to classify a

Table 6.8 ForChaos first-layer results against increasing attacks - 40 and 60 seconds window size and $\alpha = 0.1$ results

<i>Window Size (Seconds)</i>	<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>	<i>TP Rate (%)</i>	<i>FP Rate (%)</i>
40	Flooding	5 attackers	1000	500	92	0
40	Flooding	5 attackers	2000	500	92	0
40	Flooding	5 attackers	3000	500	92	0
40	Flooding	5 attackers	4000	300	86	0
40	Flooding	5 attackers	4000	400	90	0
40	Flooding	5 attackers	4000	500	92	0
60	Flooding	5 attackers	1000	200	75	0
60	Flooding	5 attackers	1000	300	80	0
60	Flooding	5 attackers	1000	400	86	0
60	Flooding	5 attackers	1000	500	89	0
60	Flooding	5 attackers	4000	200	75	0
60	Flooding	5 attackers	4000	300	80	0
60	Flooding	5 attackers	4000	400	86	0
60	Flooding	5 attackers	4000	500	89	0
60	Slow-Rate	10 apps. /each	1000	400	86	0
60	Slow-Rate	20 apps. /each	1000	500	89	0
60	Slow-Rate	10 apps. /each	4000	400	86	0
60	Slow-Rate	20 apps. /each	4000	500	89	0

malicious traffic state as a DDoS attack, especially if the network state is prone to sudden changes on incoming traffic and an increasing rate DDoS attack might seem just as another legitimate sudden change in traffic. We tested both the 40-second and the 60-second window sizes to evaluate their accuracy.

6.6.5 Results Analysis and Discussion

The results from the constant attack scenarios are presented in Table 6.8. Proceeding, we summarise the most important findings from this series of experiments.

- Both the 40-second and the 60-second window sizes achieved relatively high TP rates, equal or more than 75% in all the experiments and once again a 0% FP rate for the reasons mentioned in Chapter 6, Section 6.6.2.
- In Table 6.8 we only list the results from scenarios which the TP was not 100% and the FP was not 0% for space reasons since the results do not demonstrate anything else

apart from the fact that these specific window sizes achieve a 100% TP rate and a 0% FP rate.

- Regardless of the window size, attack type, intensity, attack start time and duration, ForChaos first-layer does not achieve any FPs mainly due to the simulated network traffic state being stable and not having major changes. On the other hand, both the 40 and the 60-second window size achieve a high TP rate across the different attack scenarios which they were evaluated. This percentage in increasing attacks was larger than the constant attacks. This was something we expected, as increasing attacks are stealthier than constant and might be missed at the beginning of their conduction.
- We observe that the 40-second window size has no issues detecting the slow-rate attacks regardless of their intensity as opposed to the constant ones. We believe that the reason for this is the following.
- In constant attacks, the attackers start attacking with all of their resources since the beginning of the attack. However, slow-rate attacks are stealthier and might be misclassified as legitimate since they make "less noise" than flooding volumetric attacks. Hence why, after a while, the 40-second window size misses the attack in all of its duration: because the difference between what it already observed with what is observing is not high. As a result, it stops classifying slow-rate attacks as a malicious activity after a while.
- In contrast, an increasing attack gradually uses of all its resources. Therefore, this gives the 40-second window size to adjust to incoming traffic and correctly classify it as something which is malicious since the traffic it monitors keeps changing from previous observations significantly.
- Despite the 40-second window size's improvement regarding the slow-rate attacks, we observe that its TP rate drops to 92% for the flooding with 5 attackers scenarios when the attack duration is 400 seconds for attack start time 1000, 2000 and 3000 seconds. It is evident that a lower intensity flooding attack when it follows an increasing approach makes the 40-second window size miss its attack start time. However, the 40-second window size immediately improves and successfully detects the rest of the attack.
- Furthermore, we observe that the 300, 400 and 500 seconds of attack scenarios' TP rate drop to 86, 90 and 92% respectively when the attack start time is increased to 4000 seconds. This indicates that more observations of legitimate traffic can momentarily negatively impact ForChaos first-layer as it fails to detect the attack from the start.

- Regarding the 60-second window size, we observe that its TP rate is decreased only in the 1000 and 4000 seconds attack start time but not on the 2000 and 3000 seconds attack start time scenarios. We believe the reason for this is that the 60-second window size at 1000 seconds made less observations and at 4000 seconds the high number of observations it made might have impacted it negatively, the same case as the 40-second window size.
- In detail, the 60-second window size TP rate drops to 75, 80, 86 and 89% for the flooding attack scenarios in 1000 and 4000 seconds attack start time. Interestingly the 60-second window size has also issues with detecting slow-rate attack scenarios again in 1000 and 4000 seconds attack start time. Evidently, the 60-second window size has issues detecting slow-rate increasing attacks regardless of their intensity (10 or 20 apps for each attacker). Evidently, a the 60-second window size does not produce better results than the 40-second window size with regards to increasing attacks.

6.6.6 ForChaos against Mixed (Flooding and Slow-Rate) Attack Experiments

Motivation

A DDoS attack's primary objective is to make the target unavailable to requests from legitimate users. An adversary, will try to "mask" their behaviour in an attempt to remain undetected for as long as they can. An interesting scenario would involve the attacker conducting a DDoS attack by using both flooding and slow-rate techniques. That can potentially mislead an IDS, as these two methods of conducting an attack are completely opposite in their way of operating. A flooding attack generates a burst amount of requests while a slow-rate attack stealthily initiates requests that are never completed.

We have to acknowledge that our assumptions are plausible provided that the adversary is to conduct a DDoS attack that is either a flooding or a slow-rate (or a combination of them) in a manner that can be detected by monitoring the features chosen. On the other hand, a sophisticated adversary can use adversarial machine learning techniques as well as monitor the network (e.g. traffic analysis and fingerprinting) to learn more about the network traffic typical behaviour. Combining the evidence from these two series of techniques they can use it to their advantage and therefore, avoid detection by our proposed system. Some examples include the following adversarial acts:

1. The adversary identifying the window size and conducting a slow-rate DDoS attack that exploits the maximum amount of waiting for a packet to arrive for request completion.

Table 6.9 ForChaos first-layer results against mixed attacks - 40 and 60 seconds window size and $\alpha = 0.1$ scenarios

<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Rate</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>
Mixed	5 attackers & 10 apps/each	Constant	1000	200,300,400,500
Mixed	5 attackers & 10 apps/each	Constant	2000	200,300,400,500
Mixed	5 attackers & 10 apps/each	Constant	3000	200,300,400,500
Mixed	5 attackers & 10 apps/each	Constant	4000	200,300,400,500

2. The adversary identifying the value of α and conducting a DDoS attack that is long enough for ForChaos first-layer to start treating the malicious behaviour as legitimate.
3. The adversary identifying the features used to detect the DDoS attacks and constructing avoidance techniques based on the acquired knowledge.
4. The adversary identifying how ForChaosLR operates and how the different layers interact with each other and disrupting this communication between them.

Adversarial machine learning is an important issues that needs to be addressed as related studies effectively supported [106], [143], [23]. However, this aspect was out of scope in the thesis.

Experiment Setup

For evaluating ForChaos, we once again constructed multiple scenarios presented in Table 6.9. In these scenarios the following variables were been used as part of the experimentation phase: duration of the attack and attack start time. We aimed to evaluate whether the type of the attack in combination with these two variables might manage to make ForChaos failing to detect the Mixed attack. We tested both the 40-second and the 60-second window sizes to evaluate their accuracy.

Results Analysis and Discussion

From the results, we observe that both of the 40 and 60-second window sizes have no issues detecting all the scenarios of mixed attacks. Due to this, we omitted a table with the results

for space reasons since the results do not illustrate anything else apart from the fact that these specific window sizes achieve a 100% TP rate and a 0% FP rate. Although, a combined attack's potentially remaining undetected was also our concern this was not proved to be the case. Our conduction of the combined attack was proved not be harder to be detected for the following reason: The total number of attackers (for both flooding and slow-rate) was 15, where 5 attackers conducted the flooding attack and 10 the slow-rate attack. Since both window sizes can detect a flooding attack with 5 attackers or a slow-rate attack with each of the 10 attackers having only 10 threads generating traffic with at least a TP rate of 75% (except in one single case of a sub-series of experiment where the TP rate was 60%), then it was expected for this series of attacks which consist of a higher in traffic magnitude to be successfully detected. In detail, based on the rates presented in Table 5.3 NS-3 DDoS attacks traffic generation parameters the average packet rate for the flooding (with 5 attackers) was 90.1 per second and for slow-rate (10 apps per attacker) was 100.9 per second. The combined attack's average packet rate was 104.6 packets per second. Therefore, the overall magnitude was greater than both attacks, hence why it was detected with a higher accuracy from ForChaos first-layer (regardless of the window size).

6.6.7 ForChaos against Flash Crowd Experiments

Motivation

Application layer flooding DDoS attacks are very similar to flash crowd traffic. In a smart home, it is unlikely for a flash crowd event to occur however, it is possible that new nodes might enter the network and generate a considerable amount of network traffic fast. We denote as "flash crowd" the event of sudden legitimate change of network traffic state due to new nodes entering the network. It is hard to distinguish flash crowd events from application layer flooding DDoS attacks because both of them generate a high amount of requests in a relatively short amount of time.

A very popular real-world case study of flash crowd denotes the web traffic observed on the web servers of FIFA World Cup 1998. In detail, the official website for FIFA World Cup 1998 (france98.com) was reported to have received a total of 1 billion requests during the world cup period. According to the statistical analysis provided, the number of incoming requests was significantly higher than normal [31]. As a result, it is a considerable challenge to maintain the overall quality of service in such cases.

The literature has a considerable number of previous studies that have focused on differentiating between flash crowd and application layer DDoS attacks. Notable examples

Table 6.10 ForChaos first-layer results against flash crowd events - 40 and 60 seconds window size and $\alpha = 0.1$ scenarios

<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Rate</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>
Flash Crowd	5,6,7 new nodes	Constant	1000	200,300,400,500
Flash Crowd	5,6,7 new nodes	Constant	2000	200,300,400,500
Flash Crowd	5,6,7 new nodes	Constant	3000	200,300,400,500
Flash Crowd	5,6,7 new nodes	Constant	4000	200,300,400,500

include [188], [252], [139], [82], [222], [247] Hence, we believe that flash crowd events is an important issue to be investigated in the context of IoT networks.

Unarguably, a flash crowd scenario is not likely to be conducted with the same intensity in an IoT network however, we believe it is important to investigate such scenarios due to the fact that a smart home-smart grid network can also experience sudden increase in traffic for legitimate reasons. Examples include a guest gathering at a smart home on the smart home side and the overuse of A/C due to a heatwave on the smart grid side.

Hence, we had to evaluate the ForChaos accuracy on such scenarios. The flash crowd scenarios are summarised in Table 6.10.

Experiment Setup

For evaluating ForChaos, we once again constructed a multiple scenarios presented in Table 6.10. In these scenarios the following variables were used as part of the experimentation phase: duration of the flash crowd event and flash crowd event start time as well as the number of new nodes entering the network. We tested both the 40-second and the 60-second window sizes to evaluate their accuracy. The main difference from a simulation point of view of the flash crowd event from the DDoS attacks is that the new nodes entering the smart home-smart grid network generate packets following the legitimate traffic packet generation distribution as the rest of the nodes in the network. In contrast, the new malicious nodes entering the network in a DDoS attack scenario follow the malicious packet generation distribution. More details can be found in Chapter 5 in Table 5.3.

Results Analysis and Discussion

In Table 6.11 we only present the results from the 40-second and 60-second window sizes that generate a FP rate. Based on the results, the 40-second window size performs sufficiently

better with regards to the 60-second window size. Proceeding, we summarise our most important findings below.

The 40-second window size can correctly classify the flash crowd event traffic as a burst amount of legitimate traffic when the flash crowd event start time is 2000, 3000 or 4000 seconds. It only produces a FP rate when the flash crowd event start time is 1000 seconds regardless of the flash crowd event duration, with the maximum FP rate being 29%. This is due to possibly ForChaos first-layer needing more observations to correctly classify incoming flash crowd traffic. This assumption is confirmed in the remaining scenarios with a later attack start time. In the 2000, 3000 and 4000 seconds attack start time the 40-second window size does not generate any FP rate.

Since the maximum amount of FP rate is 29%, we conclude that the 40-second window size allows ForChaos first-layer to adjust to the flash crowd event relatively fast after a misclassifying it as a DDoS attack.

On the other hand, the 60-second window size regardless of the flash crowd start time and duration is not able to correctly classify incoming traffic as a legitimate burst amount of traffic. As the flash crowd event start time is increased, the FP rate rises. Based on these results, we concluded that the best window size to use for ForChaos first-layer is the 40-seconds window size.

For the remaining experiments, including the flash crowd events with more nodes described below, only the 40-second window size is considered.

The results of the 40-second window size for the flash crowd events which consist of 6 nodes are presented in Table 6.12. Proceeding, we summarise our most important findings below.

In general, we observe that in this series of experiments, the 40-second window size's performance is decreased with the flash crowd events consisting 6 nodes. This was expected since the number of nodes participating in the flash crowd event has increased and therefore, the difference between observed and new traffic is greater than the flash crowd experiments consisting 5 nodes.

From the results, we observe that as the flash crowd event start time is increased, the FP rates are relatively decreased regardless of the attack duration. This indicates, that as ForChaos considers more observations it is able to classify flash crowd events correctly.

However, when the flash crowd event start time is set to 4000 seconds the FP rates are increased regardless of the flash crowd event duration.

Hence, we conclude that an increased number of observations can have a positive impact as seen from the 2000 and 3000 seconds flash crowd start time experiments.

Table 6.11 ForChaos first-layer results against flash crowd events with 5 nodes - 40 and 60 seconds window size and $\alpha = 0.1$ results

<i>Window Size (Seconds)</i>	<i>Flash Crowd Nodes</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>	<i>FP Rate (%)</i>
40	5 nodes	1000	200	20
40	5 nodes	1000	300	29
40	5 nodes	1000	400	20
40	5 nodes	1000	500	17
60	5 nodes	1000	200	75
60	5 nodes	1000	300	80
60	5 nodes	1000	400	86
60	5 nodes	1000	500	67
60	5 nodes	2000	200	50
60	5 nodes	2000	300	60
60	5 nodes	2000	400	86
60	5 nodes	2000	500	67
60	5 nodes	3000	200	100
60	5 nodes	3000	300	100
60	5 nodes	3000	400	83
60	5 nodes	3000	500	75
60	5 nodes	4000	200	75
60	5 nodes	4000	300	80
60	5 nodes	4000	400	71
60	5 nodes	4000	500	67

However, as start time is increased it can also have a negative impact, as observed from the 4000 seconds flash crowd events experiments results.

This behaviour is due to the ForChaos first-layer's functioning. A forecasting algorithm is directly dependent on previous observations to construct its predictions. The current observations monitored are going to be used for constructing the future forecasts. In our case, if an flash crowd is initiated, then the first timestamp in which the flash crowd event takes place is going to be immediately flagged as something malicious, when that is not the case.

This is due to the present network state not being similar to what the algorithm has previously observed. As a result, the error produced (forecast – actual value) is large enough to produce a positive Lyapunov exponent.

However, as the flash crowd event is in progress, more and more observations (which contain flash crowd traffic) are stored in the algorithm's list of previous observations. Hence, the forecasting algorithm at some point will start producing forecasts that are similar to

Table 6.12 ForChaos first-layer results against flash crowd events with 6 nodes - 40 seconds window size and $\alpha = 0.1$ results

<i>Window Size (Seconds)</i>	<i>Flash Crowd Nodes</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>	<i>FP Rate (%)</i>
40	6 nodes	1000	200	60
40	6 nodes	1000	300	43
40	6 nodes	1000	400	20
40	6 nodes	1000	500	25
40	6 nodes	2000	200	20
40	6 nodes	2000	300	14
40	6 nodes	2000	400	10
40	6 nodes	2000	500	8
40	6 nodes	3000	200	20
40	6 nodes	3000	300	14
40	6 nodes	3000	400	10
40	6 nodes	3000	500	8
40	6 nodes	4000	200	40
40	6 nodes	4000	300	29
40	6 nodes	4000	400	30
40	6 nodes	4000	500	17

the flash crowd traffic (since it has already monitored enough of this type of observations). Therefore, the error will not be large enough to produce a positive Lyapunov exponent anymore.

We can observe that the longer the flash crowd event lasts, the lower the FP rate becomes, regardless of the flash crowd event start time. We acknowledge two exceptions to this observations: one at the 1000th start time between the 400 and 500 seconds duration (20% and 25% respectively) and one at the 4000th start time between the 300 and 400 seconds duration (29% and 30% respectively). However, these differences cannot be considered significant and the trend between the results is valid.

The results of the 40-second window size for the flash crowd events which consist of 7 nodes are presented in Table 6.13. Proceeding, we summarise our most important findings below.

- We once again observe that in this series of experiments, the 40-second window size's performance is decreased further when compared to the equivalent results from the flash events 6 nodes experimental results.

Table 6.13 ForChaos first-layer results against flash crowd events with 7 nodes - 40 seconds window size and $\alpha = 0.1$ results

<i>Window Size (Seconds)</i>	<i>Flash Crowd Nodes</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>	<i>FP Rate (%)</i>
40	7 nodes	1000	200	80
40	7 nodes	1000	300	57
40	7 nodes	1000	400	40
40	7 nodes	1000	500	33
40	7 nodes	2000	200	80
40	7 nodes	2000	300	57
40	7 nodes	2000	400	40
40	7 nodes	2000	500	33
40	7 nodes	3000	200	80
40	7 nodes	3000	300	57
40	7 nodes	3000	400	40
40	7 nodes	3000	500	33
40	7 nodes	4000	200	80
40	7 nodes	4000	300	57
40	7 nodes	4000	400	40
40	7 nodes	4000	500	33

- Once again, this was expected since the number of nodes participating in the flash crowd event increased and therefore, the difference between observed and new traffic is greater than the flash crowd experiments consisting 5 and 6 nodes.
- From the results presented, we observe that the FP rates were significantly increased as opposed to the 5 and 6 nodes flash crowd events experiments. The results are stable regardless of the attack start time and duration.
- We also observe that the attack start time does not affect the FP rates.
- On the other hand, we observe that the FP rate is significantly decreased when the flash crowd event duration is increased.
- This observation indicates that although the 40-second window size is initially experiencing issues in correctly identifying the flash crowd traffic at the beginning, it quickly improves.

6.6.8 ForChaos against Restart Attack Experiments

Motivation

In real-world DDoS attack incidents, a DDoS attack can be re-initiated in a short period of time for various reasons such as to avoid detection by security countermeasures installed or demand a higher payment to seize their activities against their targets. Real world examples include [174] and [145].

Therefore, we aim to evaluate ForChaos first-layer against scenarios which consists of two attacks at two different distinct times. The main research questions we aimed to investigate were the following:

- Whether ForChaos first-layer detecting the first attack successfully.
- Whether ForChaos first-layer detecting the second attack successfully.
- Whether the attack type, attack start time and/or duration of the first attack impacts ForChaos first-layer's accuracy in detecting the second attack.
- Whether ForChaos first-layer correctly identifies the traffic between the two attacks as legitimate.
- Whether the attack type, attack start time and/or duration of the first attack impacts ForChaos first-layer's accuracy in classifying incoming traffic between the two attacks as legitimate.

Experimental Setup Details

For evaluating ForChaos, we generated multiple scenarios presented in Table 6.14. In these scenarios the following variables were used as part of the experimentation phase: duration of the both the first and the second attack attack and attack start times for both the first and the second attacks.

Results Analysis and Discussion

The results for the flooding attacks and for the slow-rate attacks are presented in Table 6.15. To get a better view of how ForChaos first-layer performs against this series of scenarios we include separate TP and FP rates. The two separate TP rates represent ForChaos first-layer's performance against each of the two attacks in every scenario. The two separate FP rates represent ForChaos first-layer's performance the legitimate traffic before each attack in every

Table 6.14 ForChaos first-layer results against restart attacks - 40 seconds window size and $\alpha = 0.1$ scenarios

<i>Scenario No.</i>	<i>Attack Type</i>	<i>Attack Parameters</i>	<i>Attack Start Time (Seconds)</i>	<i>Attack Duration (Seconds)</i>	<i>2nd Attack Start Time (Seconds)</i>	<i>2nd Attack Duration (Seconds)</i>
1	Flooding	10 attackers	2000	200,300,400,500	3500	200
2	Flooding	10 attackers	2000	200,300,400,500	3500	300
3	Flooding	10 attackers	2000	200,300,400,500	3500	500
4	Flooding	10 attackers	2000	200,300,400,500	4500	200
5	Flooding	10 attackers	2000	200,300,400,500	4500	300
6	Flooding	10 attackers	2000	200,300,400,500	4500	500
7	Slow-Rate	20 apps./each	2000	200,300,400,500	3500	200
8	Slow-Rate	20 apps./each	2000	200,300,400,500	3500	300
9	Slow-Rate	20 apps./each	2000	200,300,400,500	3500	500
10	Slow-Rate	20 apps./each	2000	200,300,400,500	4500	200
11	Slow-Rate	20 apps./each	2000	200,300,400,500	4500	300
12	Slow-Rate	20 apps./each	2000	200,300,400,500	4500	500

Table 6.15 ForChaos first-layer results against restart attacks (2000 seconds attack start time) - 40 seconds window size and $\alpha = 0.1$ results

<i>Scenario No.</i>	<i>1st Attack TP (%)</i>	<i>1st Attack FP (%)</i>	<i>2nd Attack TP (%)</i>	<i>2nd Attack FP (%)</i>	<i>Total TP (%)</i>	<i>Total FP (%)</i>
1a,2a,3a	100	0	100	41	100	21
1b,2b, 2b	100	0	100	57	100	30
1c,2c,3c	100	0	100	70	100	37
1d,2d,3d	100	0	100	83	100	49
4a,5a,6a	100	0	100	21	100	15
4b,5b,6b	100	0	100	30	100	20
4c,5c,6c	100	0	100	37	100	25
4d,5d,6d	100	0	100	40	100	26
7a,8a,9a	100	0	80, 88, 92	28	90, 92, 94	16
7b,8b,9b	100	0	80, 88, 92	36	93,94,95	19
7c,8c,9c	100	0	80, 88, 92	44	93,94,95	23
7d,8d,9d	92	0	80, 88, 92	50	89,91,92	25
10a,11a,12a	100	0	100	16	100	11
10b,11b,12b	100	0	100	19	100	13
10c,11c,12c	100	0	100	23	100	16
10d,11d,12d	92	0	80, 88, 92	24	94,95,96	16

scenario. We also include the final TP and FP rates which represent the total rates of them respectively for every scenario.

Based on the results presented, we include the following general conclusions:

- ForChaos first-layer is able to detect both types of flooding attacks (the scenarios in which the second attack starts at the 2500th and 3500th seconds) in all the relevant scenarios, regardless of their attack start time and duration of each.
- On the other hand, ForChaos first-layer produced satisfactory results in detecting both slow-rate attacks across the different scenarios but was not equally successful compared to the flooding ones.
- We believe this is due to the nature of the two attacks. Although both of them aim for the same objective (make the target unavailable to legitimate requests), the slow-rate DDoS attack has a lower magnitude compared to the flooding one. Hence, the TP rate across the majority of the slow-rate scenarios is lower.

We proceed to the more detailed observations and conclusions for the flooding and slow-rate attacks separately.

- ForChaos first-layer is able to detect both types of flooding attacks (the scenarios in which the second attack starts at the 2500th and 3500th seconds) in all the relevant scenarios, regardless of their attack start time and duration of each. We believe this is due to the magnitude of the attack itself. Regardless of the first and/or second attack start time and/or their duration, ForChaos first-layer is successful in detecting both attacks fully.
- However, this magnitude cause the considerable FP rate between the second and first attack in both series of scenarios. Proceeding, we explain this statement and its importance in more detail.
- The attack duration plays an important role on a forecasting algorithm's effectiveness. A forecasting algorithm is directly dependent on previous observations to construct its predictions. The current observations monitored are going to be used for constructing the future forecasts.
- In our case, if an attack is initiated, then the first timestamp in which the attack takes place is going to be immediately flagged as something malicious. This is due to the present network state not being even remotely similar to what the algorithm has previously observed. As a result, the error produced (forecast – actual value) is large enough to produce a positive Lyapunov exponent.

- As the attack is in progress, more and more observations (which contain malicious traffic) are stored in the algorithm's list of previous observations. Due to the exponential increase in resources for the attack, ForChaos first-layer is still able to successfully classify incoming traffic as malicious (in contrast to the slow-rate attack scenarios which we discuss in detail in the next section).
- However, when the attack is finished, ForChaos first-layer still falsely classifies incoming legitimate traffic as malicious. This is due to the forecasting algorithm using previous observations to make future predictions. Even if the previous observations were malicious, ForChaos first-layer takes them into consideration. When the attack is finished, present traffic has no similarities with the previous behaviour (which was malicious). As a result, the forecasting algorithm is inaccurate in its predictions since previous observations are inaccurate with present behaviour and cannot be used to accurately predict the future behaviour. Hence, the error produced between the forecast and the actual value is high enough to produce a positive Lyapunov exponent.
- As expected, the more the first attack lasts, the higher the FP rate between the first and second attack. Furthermore, we also observe that the FP rates in all scenarios from the first series (where the second attack starts at the 3500th second) are higher compared to the equivalent ones from the second series of scenarios (where the attack starts at the 4500th second). This is due to the legitimate traffic duration between the two attacks being shorter. Therefore, the FP rate is higher compared to the ones from the second series. We also observe that the second FP rate (the FP rate on the legitimate traffic between the first and the second attack) in both series of scenarios is increased when the first attack duration is also increased for the same reasons already explained.

With regards to the TP and FP rates for slow-rate scenarios:

- When comparing the two series of slow-rate scenarios, we observe that ForChaos first-layer detect the first attack fully, unless it lasts for 500 seconds. This confirms our assumptions of a forecasting algorithm facing issues when detecting attacks that last for a while and do not have a high magnitude. More specifically:
 - The attack duration plays an important role on a forecasting algorithm's effectiveness. A forecasting algorithm is directly dependent on previous observations to construct its predictions. The current observations monitored are going to be used for constructing the future forecasts.

- In our case, if an attack is initiated, then the first timestamp in which the attack takes place is going to be immediately flagged as something malicious. This is due to the present network state not being even remotely similar to what the algorithm has previously observed. As a result, the error produced (forecast – actual value) is large enough to produce a positive Lyapunov exponent.
 - However, as the attack is in progress, more and more observations (which contain malicious traffic) are stored in the algorithm's list of previous observations. Hence, the forecasting algorithm at some point will start producing forecasts that are similar to the malicious traffic (since it has already monitored enough of this type of observations). Therefore, the error will not be large enough to produce a positive Lyapunov exponent anymore.
 - That is the main reason the 40-second window size faces problems in identifying the whole duration of 500-second attacks, regardless of their attack start time, but not on attacks which consist of 200, 300 and 400 seconds duration.
 - This reason in combination with the attack being of a smaller magnitude compared to the flooding attack caused ForChaos first-layer not being able to fully detect the slow-rate attack when it lasts for 500 second.
- Examining the results obtained from the second attack (in the scenarios which the first attack lasts for 500 seconds) in the slow-rate scenarios we observe that the TP rate of the second attack is increased when the duration of the first attack is also increased. Based on the results obtained, only one attack instance is missed, specifically the first one in which the attack takes place chronologically. This is a practical problem to how a forecasting algorithm operates with regards to its window size. The attack starts at the 3500th second. When a 40-second window is used, the first time window value in which malicious traffic is present is at the 3480th second. Therefore, half of the traffic in that window size is going to be completely legitimate. Evidently, this made the detection of the attack from the start harder. Hence, the first window the attack is detected is at the 3520th second.
 - We have to point out that the case described in our previous point is more evident on the first series of slow-rate scenarios (where the second attack start at the 3500th second) but in the second series of scenarios (second attack starting at the 4500th second) this only occurs on the scenario which the first attack lasts for 500 seconds. We believe the reason for this is the longer gap between the two attacks. The forecasting algorithm has once again observed enough legitimate traffic to be able to identify the malicious one.

- With regards to the FP rates, we observe that the slow-rate FP rates are lower than their flooding equivalent ones. This is due to the slow-rate attacks' lower magnitude.
- Furthermore, as expected, the more the first attack lasts, the higher the FP rate between the first and second attack. Furthermore, we also observe that the FP rates in all scenarios from the first series (where the second attack starts at the 3500th second) are higher compared to the equivalent ones from the second series of scenarios (where the attack starts at the 4500th second). The reason is identical to the one presented in the flooding scenarios similar point already explained above.

6.6.9 Summary of ForChaos first-layer Evaluation

ForChaos first-layer accurately detects both flooding and slow-rate attacks with a high TP rate, 98.216%. ForChaos first-layer detects both flooding and slow-rate attacks, regardless of their type (constant, increasing, mixed), duration and start time. Also, even though ForChaos follows a simpler approach in detecting DDoS attacks than other algorithmic techniques such as popular machine learning algorithms, it achieves a low FP rate, 7.054%. In addition, ForChaos responds well to flash crowd events in classifying them correctly as a burst amount of legitimate traffic. The summarised results are presented and in Figure 6.5. In addition, in Table 6.16 we present the optimal parameters of ForChaos first-layer based on the entire series of experiments presented in this chapter.

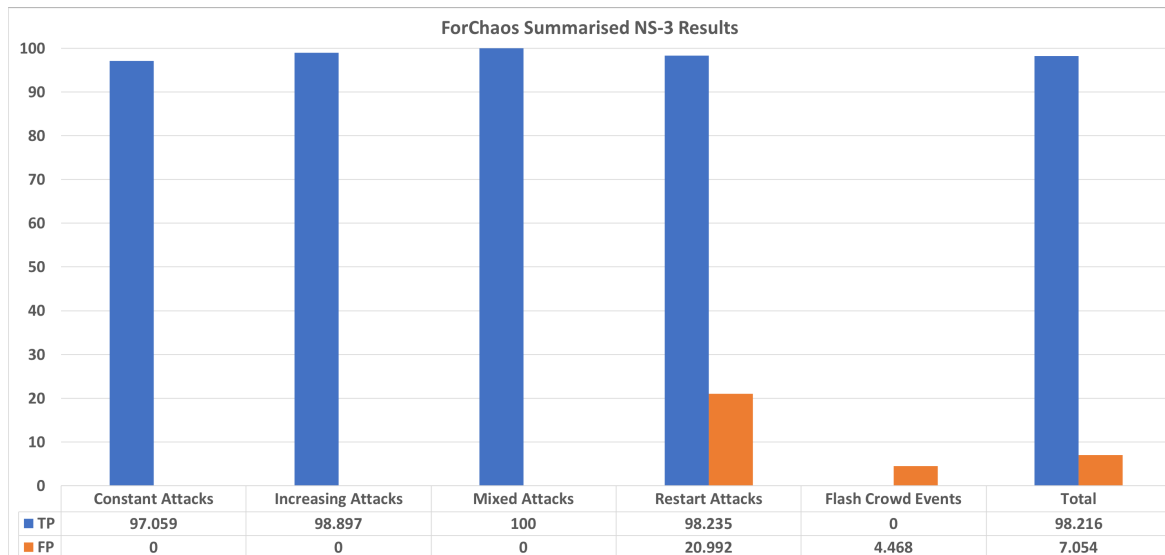


Fig. 6.5 ForChaos Summarised Results for NS-3 smart home-smart grid dataset

Table 6.16 ForChaos Variables Optimised Values for NS-3 smart home-smart grid dataset

<i>ForChaos Variable</i>	<i>Optimal Value</i>
Window Size	40 seconds
Constant α	0.1
Training Time	1000 seconds

6.7 Training Procedure of LR Second-Layer

6.7.1 Motivation

Another common problem in Machine Learning, in particular when training an algorithm, is overfitting. Overfitting means that the constructed model is dependent on the training data and is unable to generalise what it has learnt in real-world data and it is biased on the patterns it is supposed to detect.

We have to point out that different algorithms are likely to be suitable for some cyberattacks while they perform badly against others. On the other hand, previous studies demonstrated that a single sophisticated classifier (such as deep neural networks) is able to produce highly accurate results [103], [198], [83], [236] in detecting different types of attacks. However, regardless of its sophistication and/or complexity, the algorithm is likely to encounter problems with classes that consist of a very small number of examples. Examples of this issue present are described in Chapter 3, where the KDD99 dataset is used, in particular with the U2R and R2L classes. Hence, even if multiple algorithms are deployed, if the dataset is imbalanced, all of them are likely to demonstrate issues with minority classes.

To avoid situations this problem, we firstly removed duplicates from the dataset constructed and performed the a second series of the same experiments. We performed this additional series of experiments to test whether duplicated instances can negatively impact the overall training and detection rate of the algorithmic models.

The training procedure is the most important step for the correct construction of an algorithm. As presented in Chapter 4, we need to optimise the coefficients of LR second-layer. Since we monitor eight features we need to optimise nine coefficients (eight for the features plus the constant value) and the ridge value. There are multiple approaches to the training procedure that should be followed. To gain a more accurate view on how accurately LR second-layer functions, we are going to follow two different approaches: the 80/20 training and the 10-fold cross-validation which are common practices in machine learning training. The 80/20 training indicates that from the total training set 80% is going to be used for training and the remaining 20% for testing the constructed model.

The 10-fold cross validation approach belongs to the wider k-fold cross-validation technique. Cross-validation aims to validate whether the results from a statistical analysis can be generalised to another independent dataset by assessing the generated model and its ability to predict new data that have no correlation with the data it was trained with. In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (number of folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once [9], [214], [215], [42].

We chose to list 10 as the k-number based on the extensive research by [123], [125] [124]. According to their studies and research, the best value for k is 10 since it gives the best estimations. Based on their empirical evaluations, smaller or larger values give less accurate estimations.

K-cross validation is an approach to avoid the important issue of overfitting. Overfitting denotes “the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably”. An algorithmic model that is overfit ultimately contains more parameters than can be justified by the training set. This is due to having unknowingly extracted part of the natural variation (the noise) as a representation of the underlying model structure [36].

Overfitting can occur because the criteria that was used to choose the algorithmic model was different from the same criteria used to assess the suitability of the same algorithmic model. In more detail, consider the case of an algorithmic model that is selected because it performed well with the training data but it fails on maintaining this high accuracy on unseen data. The overfitting issue occurs because the algorithmic model has “memorised” the training data instead of “learning to generalise” from a trend observed during training.

In addition, there are numerous related studies in the literature (overfitting issue spreads on other areas in computer science, engineering and so on) that acknowledge this as an important issue to be solved due to the numerous implications it causes. Some examples of such studies include [36], [183], [76], [229], [237], [120], [162], [137].

Another important issue that should be avoided during training is using training set that is imbalanced. Dataset imbalance considers the issue of a dataset that its different classes are not present in equal proportions. In general, insufficient data about a particular is a common issue in many situations such as disease diagnostics, high energy physics, defective product

detection, identification of oil spills from satellite imagery, and credit card fraud detection [127]. If the proportion difference between the classes is small, then this is unlikely to be an issue. However, if this difference is large, then the algorithm will be unable to accurately identify examples from the minority class(es). This occurs due to the following reason:

Consider a two-class problem, where one of the two classes consists of a very small number of examples representing it in the training set. The (supervised) algorithm aims to minimise the cost when they falsely classify an instance (using an error function and an optimisation function). If errors for instances from both classes are treated equally, then the algorithm will develop a bias towards the class with the large number of examples. This will lead to the algorithm achieving a higher misclassification rate for the minority class incoming instances [97], [164], [232], [72], [127].

Achieving high rates of accuracy when the classes' examples are not approximately equally distributed has been a well-known and popular problem studied by many researchers across different domains [97], [164], [232], [72], [127]. The common conclusion was that the chosen algorithm's accuracy on the majority class overwhelmed the accuracy achieved on the minority class(es) during evaluation. As a result, the active research area of "learning from imbalanced data" has emerged. Dataset imbalance is an issue which has been extensively researched in the cyber security area as well [254], [45], [28], [48], [200], [111], [182], [212].

Against the imbalanced dataset issue we have three options: increase the number of instances of the minority class, undersample the majority class or change the cost function to make wrong classification of any minority instances more important than wrong classification of the majority instances. We decided to explore the first options for the following reasons: Regarding undersampling the majority class: In general, the larger the dataset is the more accurate the algorithmic model is going to be so we do not want to undersample the majority class [190]. Regarding changing the cost function, logistic regression uses an already popular and successful technique: stochastic gradient descent optimisation to optimise its cost function (cross entropy). Therefore, we would not want to interfere in this procedure. We decided to proceed with oversampling the minor class using SMOTE algorithm [1].

6.7.2 Experimental Set Up

The dataset used for training was NS-3 binary training set described in Chapter 5, Section 5.2.1. For constructing the LR-second layer we used the Weka toolbox [144]. We also used Weka's implementation of SMOTE algorithm. We chose the five nearest neighbours to be considered when creating new instances of the minor class. The training set consisted of initially 939 attack instances and 1878 normal instances. After applying the SMOTE algorithm

Table 6.17 LR second-layer optimal variables for NS-3 smart home-smart grid dataset

<i>Variable</i>	<i>Optimal Value</i>
RequestsNo Coefficient	0.016
Avg TimeBetwGetGet Coefficient	-51.8481
Avg TimeBetwOkGet Coefficient	35.9988
AverageTimeBetwOkOk Coefficient	7.3629
PacketsNo Coefficient	0.0197
Avg PacketSize Coefficient	-0.0344
AverageDataRate Coefficient	-159.4193
ParallelRequests Coefficient	0.0008
Constant Coefficient	-3.2131
Ridge Value	1.0E-8

the training set now consists of 3756 attack instances. Through Weka, we completed both of our training approaches as it provides us with the functionality to split a dataset given or perform 10-cross validation.

6.7.3 Results Analysis and Discussion

The results of LR second-layer training procedures are illustrated in Figures 6.6 and 6.7.

In general, we observe that, regardless of using the SMOTE algorithm, LR-second layer performs well in both 80/20 and 10-folds cross validation training experiments with achieving at least 98.4% TP rate and no more than 0.2% FP rate. This gives us a good indication that LR second-layer is going to perform well during the evaluation stage as the second layer of ForChaosLR multi-layer IDS.

In more detail, regarding the results from training, we observe that the results improved from applying the SMOTE algorithm in the 80/20 training approach. The training set was not heavily imbalanced however there was a significant difference between the attack and the normal class which we aimed to reduce. The TP rate was increased from 98.4 to 99.2% however, the FP rate was increased from 0 to 0.2% thus, making precision 99.7% and F-score to 99.5%. Regarding the 10-folds cross validation, we observe that the TP and FP rates remained unchanged, with a 98.9% TP rate and a 0.1% FP rate. Hence, we assume that a cross-validation training approach is not negatively affected with a small degree of dataset imbalance. Based on the training approaches followed, we present the optimal parameters of LR second-layer in Table 6.17.



Fig. 6.6 LR second-layer Training Results 80/20 for NS-3 smart home-smart grid dataset

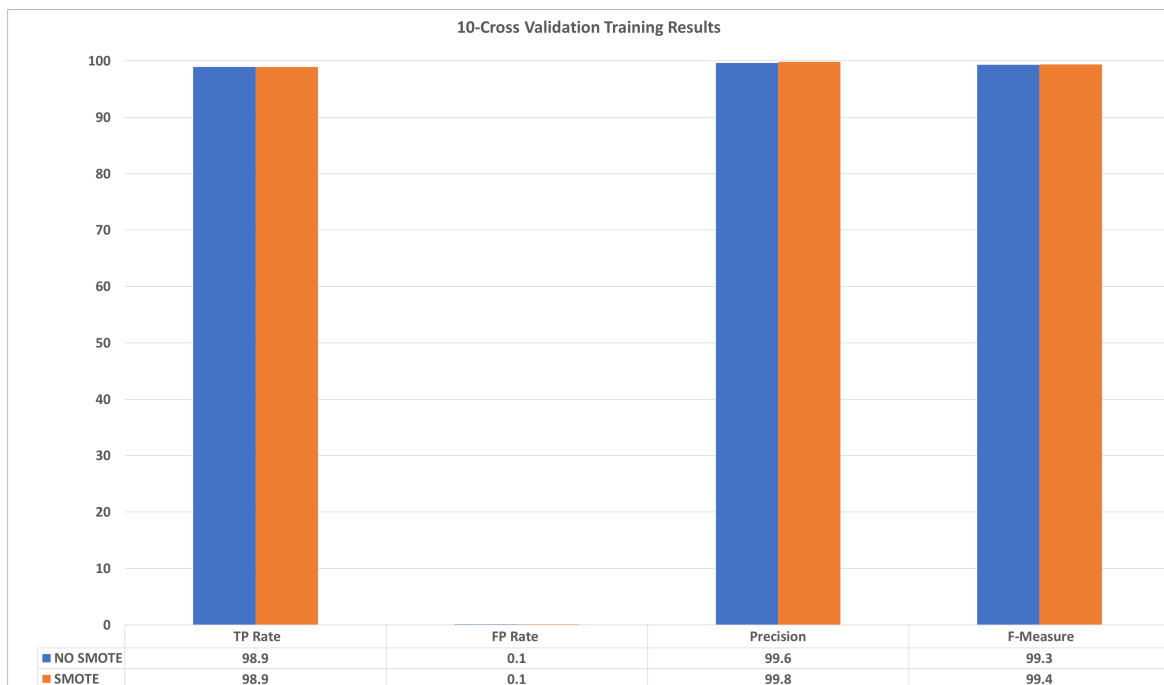


Fig. 6.7 LR second-layer Training Results 10-Cross Validation for NS-3 smart home-smart grid dataset

6.8 LR second-layer Evaluation

6.8.1 Overview

In this section, we present the results of LR second-layer against all of the scenarios conducted on ForChaos first-layer in the previous section. As described in Chapter 4, LR second-layer is activated only when ForChaos first-layer generates an alert. Hence, we collected all the generated alerts from ForChaos first-layer (both TP and FP instances) and constructed the testing set for LR second-layer. Hence, the total number of instances that were forwarded to LR second-layer was 2836 from which 2060 were attack instances and the remaining 776 were normal which were misclassified as malicious from ForChaos first-layer. To demonstrate the difference in the accuracy percentage we present how LR second-layer improves the accuracy difference from ForChaos first-layer and we summarise how the final accuracy is improved.

Experimental Set Up

For constructing and evaluating both LR second-layer and the rest of the ML algorithms we used Weka toolbox [144]. The number of attack instances evaluated for the increasing attack experiments were 538 from a total of 544, where 6 were missed from ForChaos first-layer. The number of attack instances evaluated for the mixed attack experiments were 136 all of them correctly detected and forwarded from ForChaos first-layer. The number of attack instances evaluated for the restart attack experiments were 855 from a total of 867, where 15 were missed from ForChaos first-layer and 677 normal instances. The total number of instances evaluated for the flash crowd experiments were 2216, where 99 were misclassified from ForChaos first-layer.

Results Analysis and Discussion

The results of LR second-layer are presented in Table 6.18, grouped on the type of attack. We have to highlight that these percentages are from the total number of instances forwarded from ForChaos first-layer and not the total number of instances used for the experiments. Proceeding, we summarise the most important findings. Furthermore, since the flash crowd event scenarios do not consist any malicious traffic, the TP, precision and F-score rates have been labelled as n/a as we cannot measure them.

We only assessed the FP rate to denote on whether LR-second layer falsely classified legitimate traffic as malicious.

- As expected, LR second-layer managed to get a full detection of the constant attacks for both flooding and slow-rate attacks since constant attacks are easy to be classified correctly by any machine learning algorithm when the training procedure is effective. Furthermore, as expected, LR second-layer manages to get a full detection of the mixed attacks for both flooding and slow-rate attacks. This was once again expected since mixed attacks generate enough "noise" and are easy to be classified correctly by any machine learning algorithm when the training procedure is effective.
- However, the LR second-layer's accuracy was decreased to 99.6%. This clearly indicates that increasing attacks are harder to be correctly detected. Despite the drop of TP rate from 100 to 99.6% between constant and increasing attacks, it still nearly equally high.
-
- Regarding the restart attacks, we observe that LR second-layer successfully manages to reduce the FP rate achieved from the ForChaos first-layer however, the TP rate is also reduced to 97.8%. This was once again expected, as in both the flooding and slow-rate restart attack scenarios the attacks used an increased approach. On the other hand, we conclude that LR second-layer has no issues detecting legitimate traffic between two different events of attacks that occur in a short time interval between each other as opposed to ForChaos first-layer.
- Finally, regarding the flash crowd events results, we observe, LR second-layer manages to reduce the FP rate to 56.6% achieved from the ForChaos first-layer. LR-second layer manages to classify approximately 45% of the flash crowd events as legitimate traffic, a considerable difference from the ForChaos first-layer regardless of the duration, flash crowd event start time and number of nodes taking part in the flash crowd event traffic generated.

6.9 Summary of ForChaosLR against NS-3 Dataset

- In summary, we observe from the results in Figure 6.8 that LR second-layer improves the significant FP rate achieved from the ForChaos first-layer.
- ForChaosLR multi-layer IDS decreased the FP rate from 7.054% to 0.52% after the LR second-layer is activated. Unfortunately, the TP rate was also decreased from 98.216% to 97.3%.

Table 6.18 LR-second Layer Summarised Experimental Results for NS-3 smart home-smart grid dataset

<i>Attack Type</i>	<i>TP Rate (%)</i>	<i>FP Rate (%)</i>	<i>Precision (%)</i>	<i>F-Score (%)</i>
Constant Attacks	100	0	100	100
Increasing Attacks	99.6	0	100	99.8
Mixed Attacks	100	0	100	100
Restart Attacks	98.0	0.0	100	99.0
Flash Crowd Events	n/a	56.6	n/a	n/a
Total	99.1	7.3	97.3	98.2

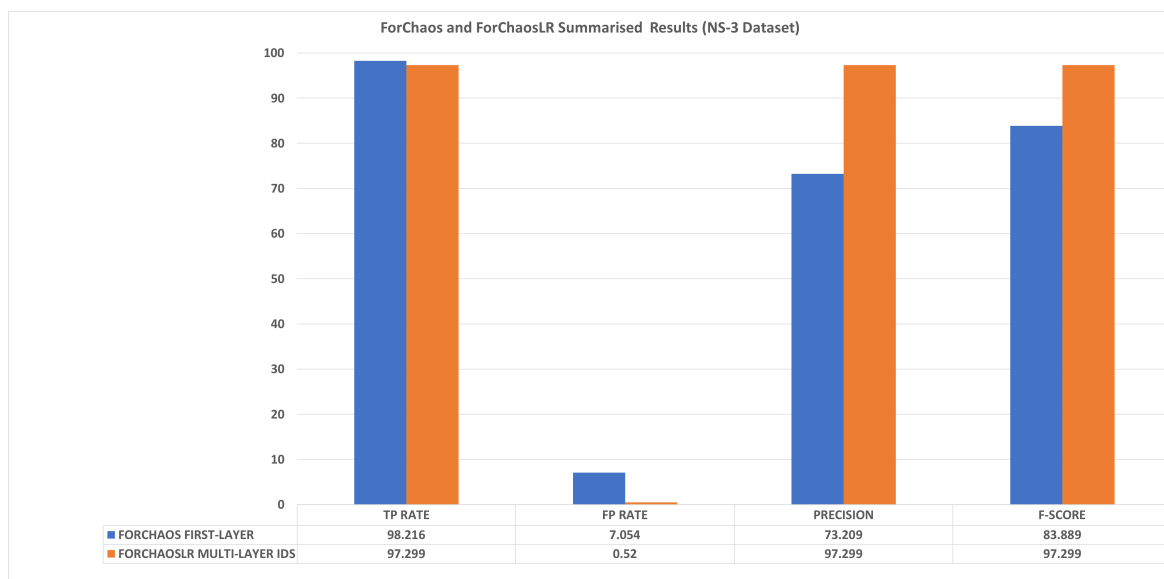


Fig. 6.8 ForChaos first-layer and ForChaosLR NS-3 Binary Summarised Dataset Results

- To investigate on whether the difference is significantly different we constructed a two sample t-test. The result of the two sample t-test was 1.149. With 6 D.F and $p=0.05$ the corresponding value was 4.32. Since $1.149 < 4.32$, we can accept the null hypothesis, that the two sample means are not significantly different from each other.
- In addition, the TP rate confidence interval of LR second-layer for every scenario is 99.3427 ± 0.1494 (with a 95% confidence level).
- Although the decrease is not a big difference we assume this is the case for a number of reasons explained below.
- In general, it was expected from ForChaos first-layer to achieve a rather significant FP rate since it was used as an anomaly-based detection algorithm. Anomaly-based algorithms are constructed with only what is considered "normal behaviour" and classify whatever does not fall between these boundaries as malicious. However, legitimate behaviour can have multiple variations that are not similar to each other. Hence why ForChaos first-layer was ineffective against flash crowd event scenarios.
- For the reasons mentioned above, an anomaly-based detection algorithm can classify new incoming behaviour as malicious for the sole reason of it not being similar to past seen legitimate behaviour. LR second layer, specifically the logistic regression model, proved to be particularly effective in classifying normal behaviour correctly forwarded falsely from the ForChaos first-layer. Our findings agree with the studies presented in Chapter 3, specifically [11], [107], [184], [221], [11], [107] and [184].

6.10 Evaluation of ForChaosLR against popular machine learning algorithms

6.10.1 Overview

To evaluate the accuracy of ForChaosLR multi-layer IDS in further, we used other popular machine learning to detect application layer flooding and slow-rate attacks. Specifically, the algorithms used were the following: Bayesian networks (BN), naive Bayesian (NB), K-nearest neighbour (KNN), support vector machine (SVM), decision tree (DT), random forest (RF) and artificial neural networks with the multi-layer perceptron architecture (MLP).

6.10.2 Experimental Set Up

For the MLP, KNN and SVM algorithms the following key parameters were tweaked in an attempt to examine whether their results can be potentially improved: learning rate for MLP, nearest neighbour along with distance and weighting used for KNN and kernel type for SVM. The specific parameters used for every algorithm can be found in more detail in Table 6.19. The implementations of Weka [144] were used for all the algorithms. Regarding training, the training set used was the same as the one used for LR second-layer. Regarding testing, the test set used was the same as the one ForChaosLR was evaluated with.

Table 6.19 Machine Learning algorithms Parameters

<i>Algorithm Name</i>	<i>Parameter Details</i>
Bayesian Networks (BN)	Estimator: Simple Estimator Search Algorithm: K2 (default values)
Naive Bayes (NB)	User Kernel Estimator: False (default values)
Multi-Layer Perceptron (MLP(0.3))	Learning Rate: 0.3 (default value)
Multi-Layer Perceptron(MLP(0.5))	Learning Rate: 0.5
SVM(Linear)	Kernel: Linear (default value)
SVM(Quadratic)	Kernel: Quadratic
SVM(Cubic)	Kernel: Cubic
SVM(RBF)	Kernel: RBF (Radial Basis Function)
KNN(1)	Nearest Neighbours: 1 Nearest Neighbour Search Algorithm: Linear NN Search Distance used: Euclidean, Chebyshev, Manhattan, Filtered Distance Weighting: None(default value)
KNN(3)	Nearest Neighbours: 3 Nearest Neighbour Search Algorithm: Linear NN Search Distance used: Euclidean, Chebyshev, Manhattan, Filtered Distance Weighting: None(default value)
Decision Tree (DT)	SubTree Raising: True Unpruned: False Use Laplace: False (default values)
Random Forest(RF)	Break Trees Randomly: False

6.10.3 Results Analysis and Discussion

In Figures 6.9, 6.10, 6.11 and 6.12 we present the results for NS-3 binary dataset. In general, we observe that all of the ML algorithms achieved satisfactory results, with the TP rate being

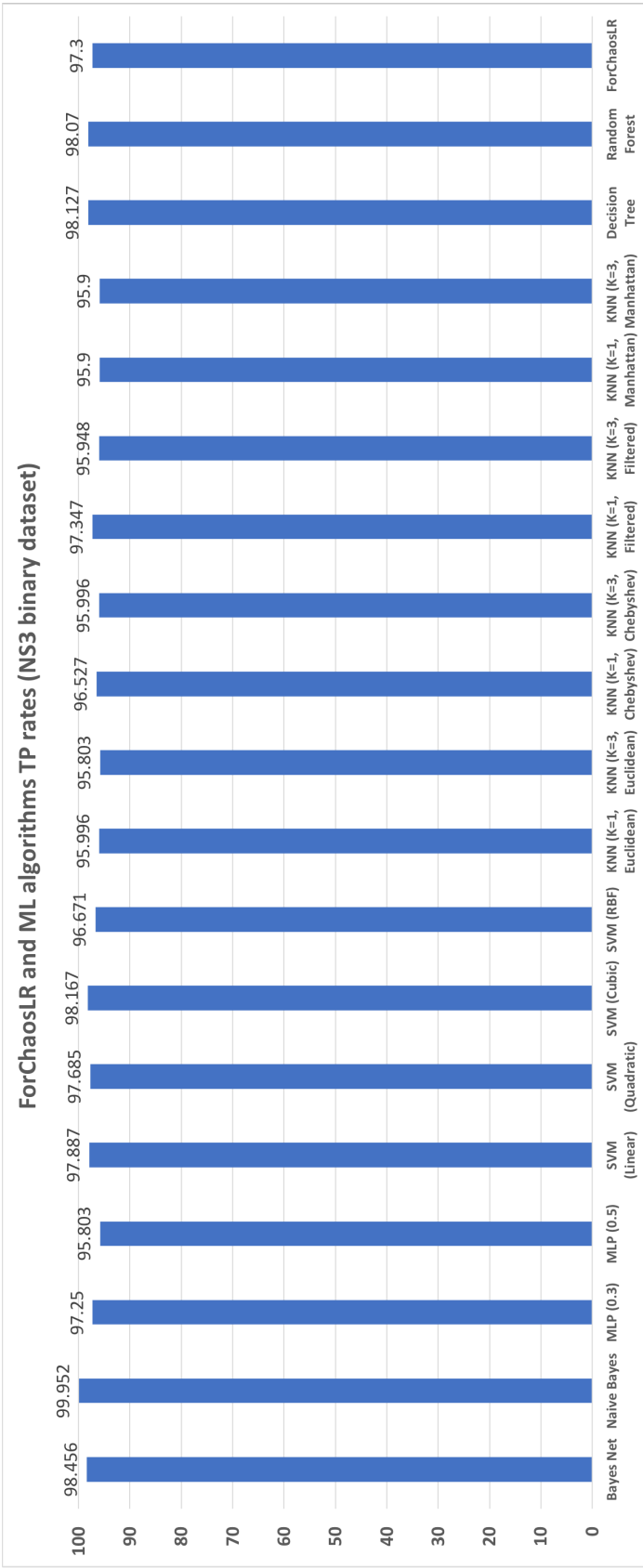


Fig. 6.9 ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (TP rates)

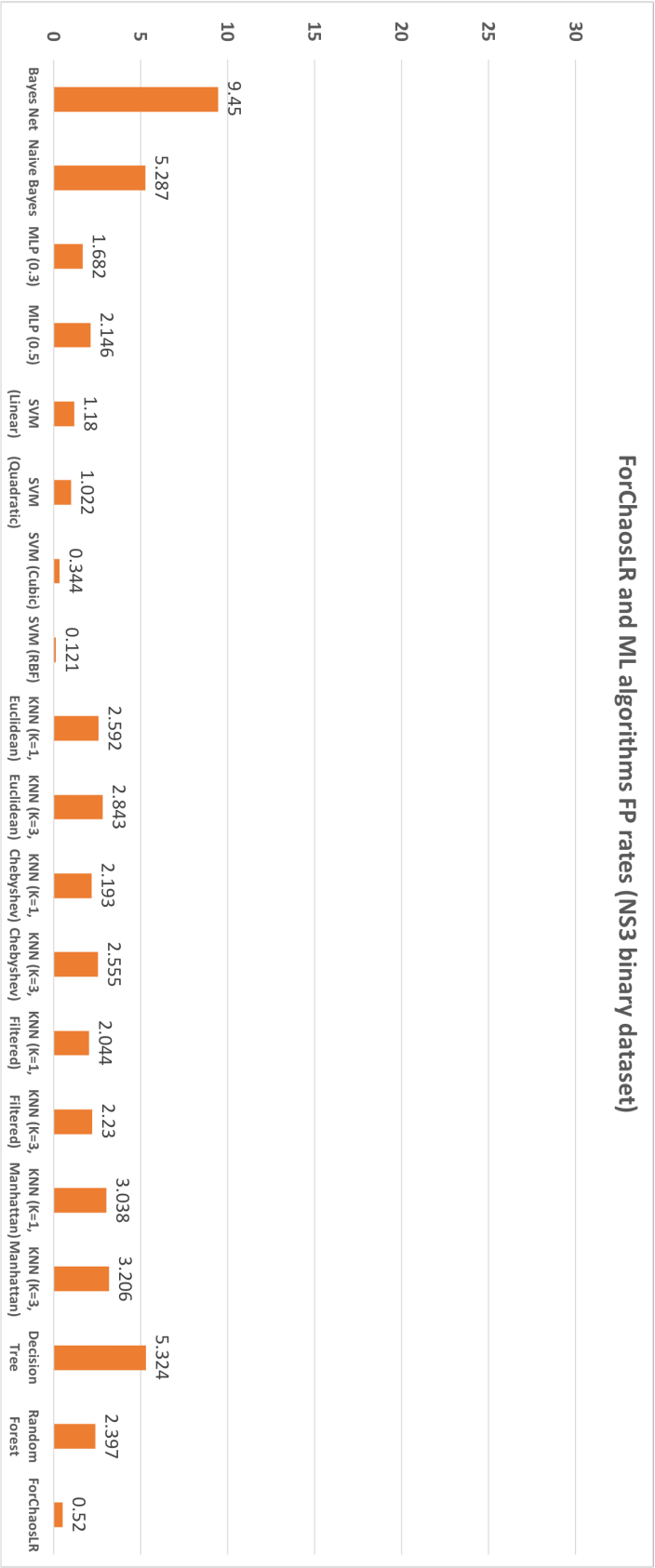


Fig. 6.10 ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (FP rates)

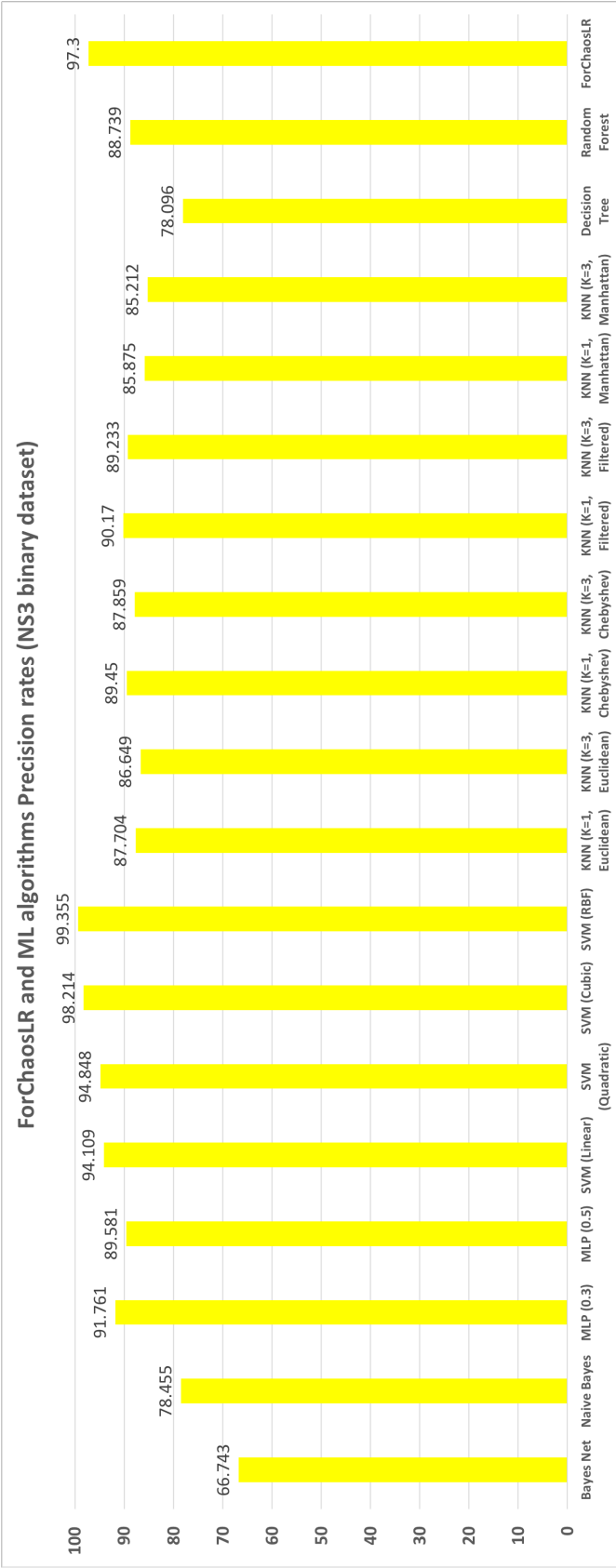


Fig. 6.11 ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (Precision rates)

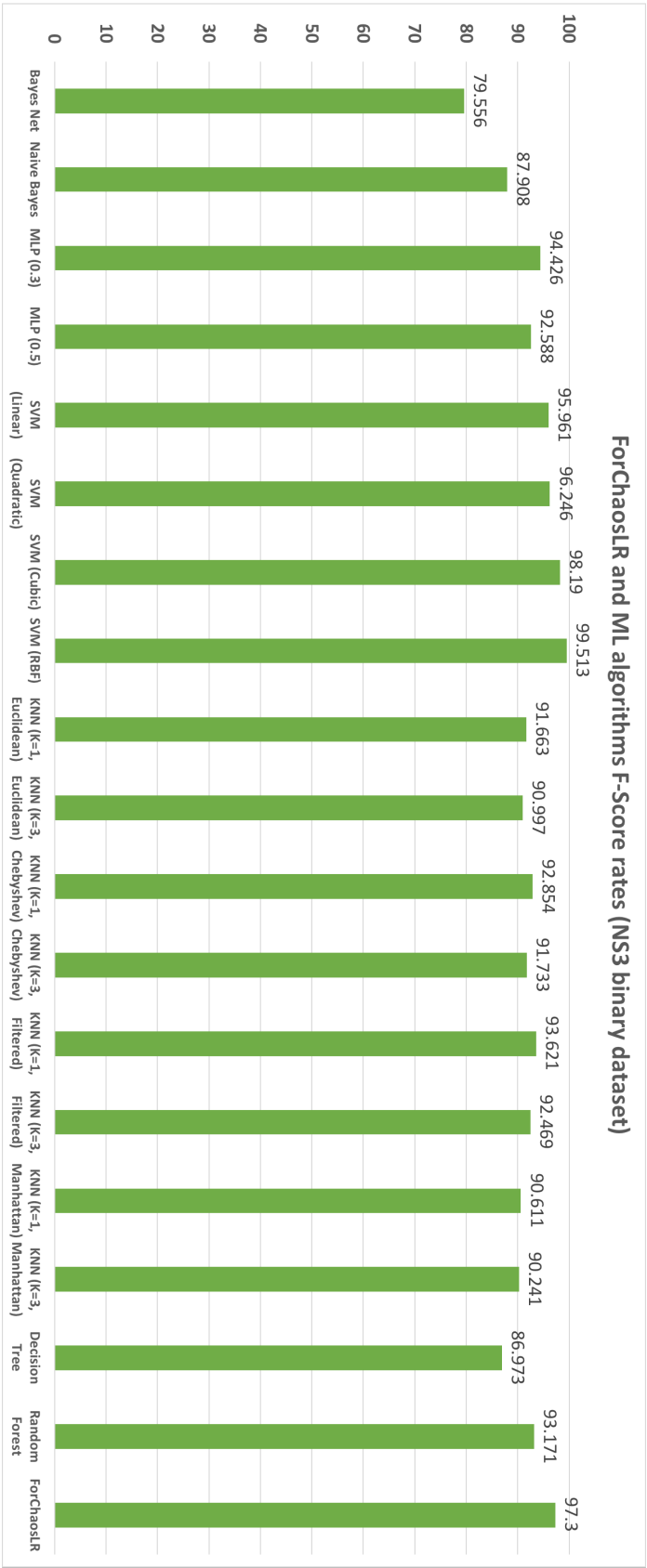


Fig. 6.12 ForChaosLR multi-layer and ML algorithms NS-3 smart home-smart grid dataset Results (F-Score rates)

95.900% or more and the FP rate being no higher than 9.450%. Proceeding, we summarise our conclusions below.

Bayesian Algorithms :

- Examining the probabilistic algorithms we note that in this round of experiments Bayesian networks algorithm's TP rate is 98.456% while naive Bayes algorithm's TP rate being 99.952% with an approximate 1.5% difference.
- Furthermore, naive Bayes algorithm's FP rate is 5.287%, approximately 4.2% lower than Bayesian network algorithm's FP rate, 9.450%. This indicates that although naive Bayes algorithm is simplistic in terms of algorithmic model construction, it is more accurate both in TP and FP rates.

Neural Network Algorithms:

- Examining the neural network MLP results, we observe that MLP (0.3) achieved a better TP rate, with 97.250%, than MLP (0.5) which generated a 95.803% TP rate, an approximate difference of 1.5%.
- In addition, MLP(0.5) achieved a higher FP rate than MLP(0.3), with 2.146% and 1.682% respectively. This gives us a clear view that the learning rate is a parameter that can significantly impact the TP rate as well as the FP rate but to a lesser degree.

SVM Algorithms :

- Examining the SVM algorithms results, we observe that the TP rates across all different kernel versions of the SVM algorithm produce similar TP and FP rates.
- In detail, we observe that SVM cubic kernel achieved the highest TP rate 98.167%, an expected result since the kernel used is more complex which makes it suitable for effectively classifying flooding, slow-rate and legitimate traffic accurately.
- SVM(RBF) achieved the lowest TP rate compared too the SVM polynomial kernels.
- On the other hand, the RBF kernel SVM algorithm's TP rate is considerably lower, approximately 1.5%. In contrast with the TP rates, RBF kernel SVM algorithm generates the lowest FP rate, with only 0.121%, lower by approximately 0.2% than the cubic kernel SVM algorithm but approximately 1% lower than SVM linear and quadratic kernel algorithms.

KNN Algorithms:

- Examining the KNN algorithms results, we observe that all of the KNN versions that used $K=1$ have lower FP rates regardless of the distance measure used. This indicates that an increase in K number has a negative impact on FP rates however, the differences between the rates is not significant.
- In more detail, filtered distance achieved the best results as opposed to the other distance measures used that do not use a filter beforehand. Filtered distance is different from the other distance measures because it applies a filter before calling the distance measure. The filter we selected is called "Random Projection" and the distance measure selected is Euclidean distance. The "Random Projection" filter aims to decrease the dimensionality of the data.
- To achieve this, it projects the data into a lower dimensional subspace by using a random matrix however it does not compromise the variation of the data used. As we observe from the results, the usage of filter that decreases the dimensionality of the data, produces significantly higher TP rate than the other distance measures with a 97.347% TP rate.
- By examining the individual results each combination of distance with number of K , we can observe that excluding the filtered distance, the highest TP rate generated by the Chebyshev distance with $K=1$ having a 96.527% TP rate and the least TP rate generated from Manhattan distance with $K=1$ and $K=3$ having a 95.900% TP rate.
- By excluding the filtered distance results, when comparing the results across the three different distance measures, Chebyshev comes first, followed by Euclidean and then Manhattan distance. We believe Chebyshev produces the best results out of the three because it takes the maximum distance of comparing each point of the node to be classified with the equivalent of the nearest neighbour node instead of adding the results from subtracting each point of the two vectors as Euclidean and Manhattan distance measures do.
- Excluding the filtered distance KNN results, all KNN different distance measure types produce lower TP and higher FP rates when the K is increased however their difference is not significant.
- Regarding the FP rates, filtered distance achieves the lowest FP rate (2.044% and 2.23%) and the Manhattan the highest (3.038% and 3.206%) followed by Euclidean (2.592% and 2.843%) and Chebyshev (2.193% and 2.555%) distances. We have to point out that the differences across the FP rates are not significant though.

Tree Algorithms:

- Examining the tree-based algorithms results, we observe that the decision tree algorithm produces better TP rate 98.127% than the random forest algorithm with 98.070%, however their difference is not significant.
- In contrast, when comparing their FP rates we note that decision tree algorithm's FP rate is more than doubled, 5.324% compared to Random Forest algorithm's, 2.397%.
- Random forest is an extension of the decision tree algorithm, designed to improve the former's results by dividing the training set and from the sub-training sets constructing multiple trees.
- Random forest algorithm classifies incoming instances based on majority voting to prevent overfitting and potentially produce more accurate results.
- This is reflected in the results, as random forest algorithm clearly has an advantage in classifying legitimate traffic correctly as opposed to decision tree algorithm.

ForChaosLR results and comparison with ML algorithms:

- Examining ForChaos multi-layer IDS's results, we observe that it achieves a high TP rate, higher than the KNN algorithms, except KNN with Filtered distance, MLP neural networks and SVM RBF. It appears to be that the approximating probabilistic approach followed by logistic regression has a clear advantage over KNN distance-based algorithms, SVM(RBF) and the heuristics followed by MLP neural networks.
- Regarding FP rates, we observe that ForChaos multi-layer IDS achieved one of the lowest FP rates when compared to the ML algorithms with only 0.520%. Only SVM(Cubic) and SVM(RBF) achieve a lower FP rate than ForChaosLR. Hence, we observe that ForChaosLR has a clear advantage on FP rates over Bayesian algorithms (both Bayesian networks and naive Bayesian), tree-based approaches, neural networks and KNN techniques. We believe that the approximating probabilistic nature of logistic regression has a clear advantage over the other algorithms that use different approaches for their model construction.
- Based on the results, we observe that the only algorithm that achieved higher TP rate and lower FP rate is SVM(Cubic). It is clear that the cubic kernel used in the construction of the hyperplane is a successful method compared to the rest of the ML algorithms, including ForChaosLR. The higher degree (3rd) of the polynomial kernel

is a more complex approach to construct a highly accurate hyperplane, able to correctly differentiate between malicious and legitimate traffic. This is evident when comparing the polynomial degree SVM algorithms with each other. When the degree is increased the overall accuracy is also improved.

- With regards to the ML algorithms that have either higher FP rate or lower TP rate than ForChaosLR we make the following conclusion; in general, such complex methods are more accurate after being trained with a very large in-size training set. Hence, we believe that their TP rate is likely to be increased and their FP rate is likely to be reduced when the training set size is substantially increased. On the other hand, ForChaosLR produces satisfactory results with the training procedure as it is.

6.11 Summary

In this chapter, we presented all of the experimental results for ForChaosLR multi-layer IDS using the generated NS-3 binary dataset generated through our NS-3 smart home-smart grid network simulation, presented in Chapter 5, Section 5.4.

Both ForChaos and LR layers were trained so the most optimal variables are found. The appropriate testing procedures were followed for both ForChaos and LR layers through a plethora of different scenarios to assess both their accuracy in detecting flooding and slow-rate attacks under various variables such as attack duration, attack start time, intensity and whether the attack re-starts as well as their sensitivity to sudden legitimate changes in traffic such as flash crowd events.

Furthermore, different ML algorithms were evaluated using the NS-3 dataset so their results could be compared with the results generated from ForChaosLR multi-layer IDS. ForChaosLR multi-layer IDS achieves a 97.3% TP rate, a result when compared to the ML algorithms indicates that it generates a higher TP rate than many of the ML algorithms and the difference of ForChaosLR from the rest of them is no more than approximately 2.7%. In more detail, ForChaosLR achieves a higher TP rate from all KNN algorithms (except KNN Filtered and K=3), MLP(0.5) and SVM(RBF). Regarding FP rates, ForChaosLR generates a 0.520% rate, which is only higher than SVM(RBF) and SVM(Cubic).

Chapter 7

ForChaosLR Accuracy Evaluation using Real-World Datasets

7.1 Overview

In the previous chapter we focused on the effective training of both layers of ForChaosLR using the NS-3 smart home-smart grid binary dataset to evaluate whether ForChaosLR can detect application layer flooding and slow-rate DDoS attacks in the smart home-smart grid network. In this chapter, we proceed with the in-depth experimental analysis of ForChaosLR multi-layer IDS using the ISCXIDS 2012 (Section 5.2.2), binary CICIDS 2017 (Section 5.2.2) and the smart home (Section 5.2.2) datasets to complete the following objectives.

- Evaluate ForChaosLR multi-layer IDS's ability to detect both flooding and slow-rate attacks efficiently in traditional ICT networks, thus making a suitable lightweight solution for ICT networks using the ISCXIDS 2012 and CICIDS 2017 datasets.
- Empirically evaluate the validity of DDoS flooding and slow-rate attacks simulations in NS-3 binary dataset, through the similar TP rates generated.
- Investigate whether ForChaosLR multi-layer IDS is able to detect other types of application layer DDoS attacks (except from flooding and slow-rate slowloris) on the application layer such as Goldeneye, SlowHTTPTest as well as HULK flooding attack [92] besides basic flooding attack using the CICIDS 2017 dataset.
- Evaluate ForChaosLR multi-layer IDS's ability to correctly assess real-world smart home traffic, using the smart home dataset.

- Prove the validity of smart home-smart grid network traffic simulations in NS-3 binary dataset, through the similar FP rates generated.

To assess ForChaosLR multi-layer IDS's accuracy we used multiple evaluation metrics including as true positive rate (TP rate), false positive rate (FP rate), precision and F-score. We also compared ForChaosLR multi-layer IDS against a variety of different machine learning (ML) algorithms. More details regarding the datasets can be found in Chapter 5 Section 5.2.2. The next sections are organised and briefly described below.

1. Evaluation of ForChaosLR against ISCXIDS 2012 and CICIDS 2017 Datasets: This section presents and describes in detail the evaluation of ForChaosLR multi-layer IDS using the ISCXIDS 2012 and the CICIDS 2017 datasets. In detail:
 - (a) Motivation: The reasoning for conducting this series of experiments using the ISCXIDS 2012 and CICIDS 2017 datasets is presented and explained.
 - (b) Experimental Setup Details: The relevant details regarding the construction of the experiments are presented, specifically the generated values for the parameters of ForChaosLR multi-layer IDS during training in Table 6.2.
 - (c) Results Analysis and Discussion: The evaluation results of ForChaosLR multi-layer IDS are presented and discussed by using the TP and FP rates as well as precision and F-score.
2. Evaluation of ForChaosLR against smart home datasets: This section presents and describes in detail the evaluation of ForChaosLR multi-layer IDS using smart home datasets. In detail:
 - (a) Motivation: The reasoning for conducting this series of experiments using the smart home datasets is presented and explained.
 - (b) Experimental Setup Details: The relevant details regarding the construction of the experiments are presented, specifically the generated values for the parameters of ForChaosLR multi-layer IDS during training in Table 6.2.
 - (c) Results Analysis and Discussion: The evaluation results of ForChaosLR multi-layer IDS are presented and discussed by using the TP and FP rates as well as precision and F-score.

For every dataset used, appropriate training procedure was followed where the optimal values for the parameters present in our algorithmic design presented in Chapter 4 along with

their respective descriptions. A summary of these variables is presented in Table 6.2 along with what layer they belong to. Referring to the optimal values, for the ForChaos first-layer we denote α , window size w and training time. For the LR second-layer we denote the coefficients and ridge value of the logistic regression function corresponding to each feature in the function that combined together give the most accurate results. Furthermore, due to the algorithmic nature of the two layers, different training procedures need to be followed for each one.

ForChaos first-layer needs to be trained with only real-time data for a period of time so it can gather enough data for its forecasting model to start making accurate predictions. During this period of training, α , the window size w and training time are to be decided. On the other hand, LR second-layer needs a considerably large dataset with both legitimate and DDoS instances. A large dataset assists into finding the coefficients for the logistic regression model that give the lowest error rate and make the most accurate predictions for incoming data.

7.2 Evaluation of ForChaosLR against ISCXIDS 2012 and CICIDS 2017 Datasets

7.2.1 Motivation

Although the NS-3 smart home-smart grid dataset was constructed by considering all the latest specification and standards and simulating the protocols according to their respective official manuals the data are still synthetic and not real-world. To avoid any issues of bias in the results we aim to use other data from real-time environments which were necessarily captured in a smart home and a smart grid environment as our objective is to evaluate ForChaosLR multi-layer IDS's accuracy in detecting flooding and slow-rate application DDoS attacks.

Hence, we used two datasets from the Canadian Institute for Cybersecurity of the University of New Brunswick, namely ISCXIDS 2012 described in Chapter 5, Section 5.2.2 and CICIDS 2017 described in Chapter 5, Section 5.2.2. Both of these datasets contain realistic network traffic which also includes of various application layer DDoS attacks. We also aim to evaluate ForChaosLR multi-layer IDS's potential capability in detecting other types of application layer DDoS attacks beyond flooding and slow-rate.

Both of these datasets contain realistic network and traffic which consists both normal and malicious traffic. Amongst the attacks simulated, they contain application layer DDoS attacks. Specifically they contained flooding generated with LOIC/HOIC, HULK and GoldenEye [92] and slow-rate attacks generated with Slowloris and SlowHTTPTest [92].

Table 7.1 LR second-layer variables ISCXIDS 2012

<i>Variable</i>	<i>Optimal Value</i>
RequestsNo Coefficient	0.0005
Avg TimeBetwGetGet Coefficient	-28.0524
Avg TimeBetwOkGet Coefficient	-9.6719
AverageTimeBetwOkOk Coefficient	-24.1526
PacketsNo Coefficient	0.0004
Avg PacketSize Coefficient	-0.0187
AverageDataRate Coefficient	-159.4193
ParallelRequests Coefficient	0
Constant Coefficient	-12.7164
Ridge Value	1.0E-8

Table 7.2 LR second-layer variables CICIDS2017

<i>Variable</i>	<i>Optimal Value</i>
RequestsNo Coefficient	0.0104
Avg TimeBetwGetGet Coefficient	3694.2396
Avg TimeBetwOkGet Coefficient	0
AverageTimeBetwOkOk Coefficient	0
PacketsNo Coefficient	-0.0021
Avg PacketSize Coefficient	0.0008
AverageDataRate Coefficient	4.3174
ParallelRequests Coefficient	0
Constant Coefficient	-9.3706
Ridge Value	1.0E-8

The ISCXIDS 2012 training set contained 2548 attack and 3712 normal instances. The ISCXIDS 2012 testing set contained 94 attack and 2426 normal instances. The CICIDS 2017 training set contained 1862 attack and 1119 normal instances. The CICIDS 2017 testing set contained 130 attack and 295 normal instances.

7.2.2 Experimental Setup Details

For training LR second-layer and the ML algorithms we used the ISCXIDS 2012 training set, described in Chapter 5, Section 5.2.2. The logistic regression function coefficients and ridge value for LR-second layer are presented in Table 7.1. For training LR second-layer and the ML algorithms we used the CICIDS 2017 training set, described in Chapter 5, Section 5.2.2. The logistic regression coefficients and ridge value for LR-second layer are presented in Table 7.2.

7.2.3 Results Analysis and Discussion of ISCXIDS 2012

To get a more clear and detailed view of the results for the ISCXIDS 2012 testing set and what improvements were made to increase the accuracy of ForChaosLR multi-layer IDS we present the individual results from the flooding DDoS attack day and the normal traffic day.

7.2.4 ForChaosLR ISCXIDS 2012 Flooding Day Results

According to Figure 7.1, we observe that ForChaosLR multi-layer IDS is able to fully detect the flooding attack in all of its duration. However, it achieves a total of 9.061% FP rate.

Examining the results, we observe that both ForChaos first-layer and LR second-layer have no issues in detecting the flooding attack. This is due to the attack's magnitude as it is demonstrated in Figure 7.1, where the rate of incoming application layer packets is displayed per second, extracted from Wireshark.

However, despite ForChaos first-layer's accuracy, it achieves a high FP rate, specifically 99.838%, after the attack is conducted. This is evident in the Restart experiments presented in Chapter 6, Section 6.6.8, especially the flooding scenarios.

We conclude that the larger the magnitude of a DDoS attack, the longer ForChaos first-layer needs to start producing accurate results with regards to future incoming traffic. This is a problem due to the way ForChaos' first-layer operates, as it is dependent upon previous observations.

Therefore, if an attack large in magnitude is conducted for a lengthy amount of time, ForChaos first-layer will fail to classify incoming legitimate traffic as normal. This is due to a large amount of time it was observed only attack traffic and keep producing falsely positive Lyapunov exponents as it is indeed shown in Figure 7.2. The Lyapunov exponents should have stopped being positive after the attack was finished.

Hence we conclude that when a DDoS attack with high-magnitude occurs, ForChaos first-layer should be re-trained after an attack is finished, so the FP rate is minimised. During that period of time, LR second-layer can act as the first line of defence with regards to monitoring incoming traffic until the retraining procedure of ForChaos first-layer is finished.

As previous related studies that use forecasting algorithms [225], [121], [16], acknowledge and support that such methods need to be retrained at some point so they can continue producing accurate predictions. The reason for this is that forecasting algorithms are directly dependent on previous observations to make their predictions. Since network traffic contains natural, and possibly frequent, variations, its current state might be different from previous states. Hence, the forecasting algorithm needs to be retrained.

Forecasting algorithms are directly dependent on previous observations to make their predictions. Since network traffic contains natural, and possibly frequent, variations, its current state might be different from previous states. Hence, the forecasting algorithm needs to be retrained.

To evaluate this assumption we retrained ForChaos first-layer after the attack is finished. We used the same parameters as in previous experiments, approximately 1000 seconds of training time (25052th-26173th second), $\alpha = 0.1$ and a window size of 40 seconds. This set of parameters were proved sufficient as we observed from Figure 7.3.

ForChaos first-layer achieves no FP rate when re-trained after the flooding attack is finished. This is a very reassuring improvement but should be taken with caution as ForChaos is likely to produce a FP rate as time goes by. As a result, when re-training of ForChaos is conducted, ForChaosLR multi-layer IDS achieves a 100% TP rate and a 0% FP rate for this June 15 flooding event as shown in Figure 7.4.

7.2.5 ForChaosLR ISCXIDS 2012 Normal Day Results

Proceeding to the ForChaosLR multi-layer IDS results from the ISCXIDS 2012 normal day we observe that ForChaosLR multi-layer IDS was robust and accurate in classifying incoming traffic as legitimate as shown in Figure 7.5.

Examining the results, we observe that ForChaos first-layer can initially correctly classify incoming traffic as legitimate. The Lyapunov exponents remain negative with however, multiple temporary spikes of positive Lyapunov exponents due to the high instability of traffic. This observation is illustrated in Figure 7.6 where the incoming number of packets is illustrated using Wireshark. Despite this, we observe that after approximately 19000 seconds (approximately 5.3 hours) the number of positive Lyapunov exponents is increased exponentially. This observation is in Figure 7.7 and remains positive for the majority of time. As a result, ForChaos first-layer achieves a very high FP rate, specifically 84.372%.

We believe that is because of the following reason: Traditional ICT networks can produce traffic that is less stable due to legitimate temporary spikes on multiple occasions throughout a period of time. The multiple changes in incoming traffic make it harder for ForChaos first-layer to make accurate classification of traffic as the new incoming traffic. Although legitimate, these legitimate spikes are vastly different from what ForChaos first-layer was previously observed.

In contrast, smart home-smart grid networks are unlikely to have frequent sudden burst amount of traffic in a relatively short amount of time. This explains the difference between the results of ForChaos first-layer in the NS-3 binary dataset and the results of ForChaos first-layer. Even though, LR second-layer reduces the very high FP rate to 0%, we

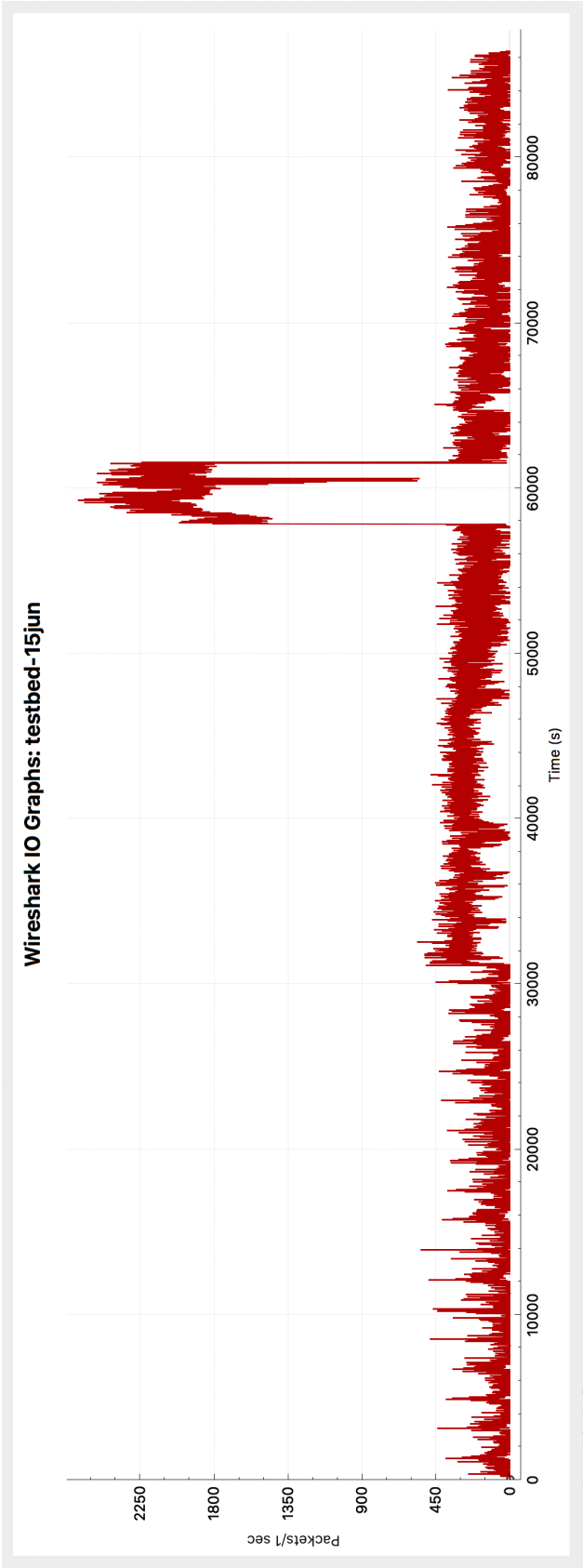


Fig. 7.1 ISCXIDS2012 15th June Flooding Attack Packet Arrival Time/sec

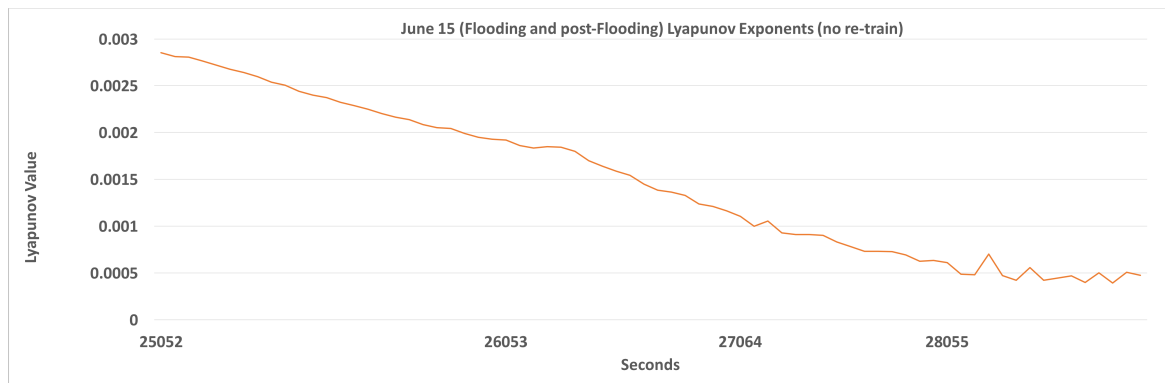


Fig. 7.2 ISCXIDS2012 15th June Flooding Attack Lyapunov Exponents Behaviour (no re-train)

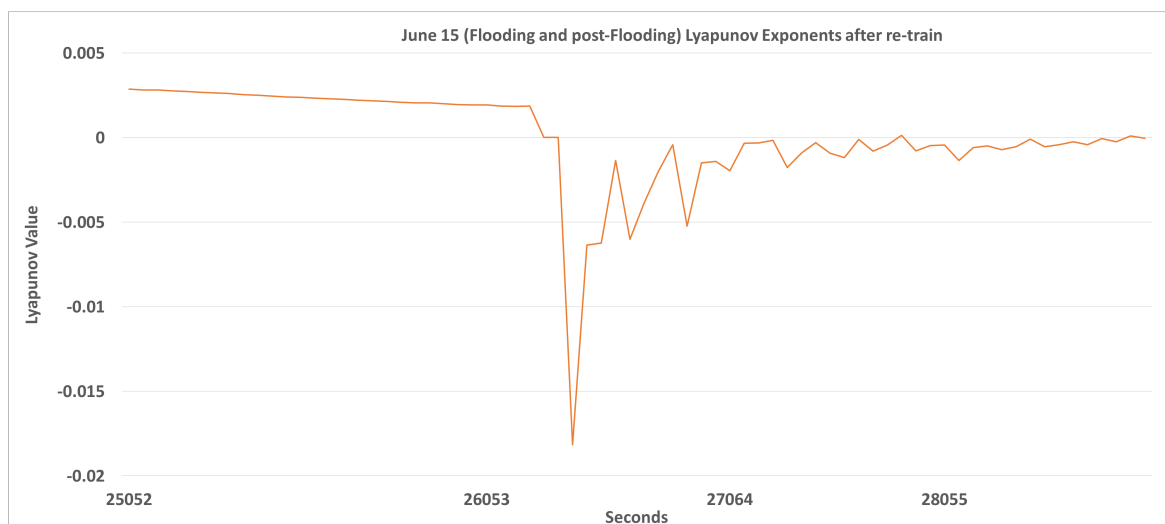


Fig. 7.3 ISCXIDS2012 15th June Flooding Attack Lyapunov Exponents Behaviour (after re-train)

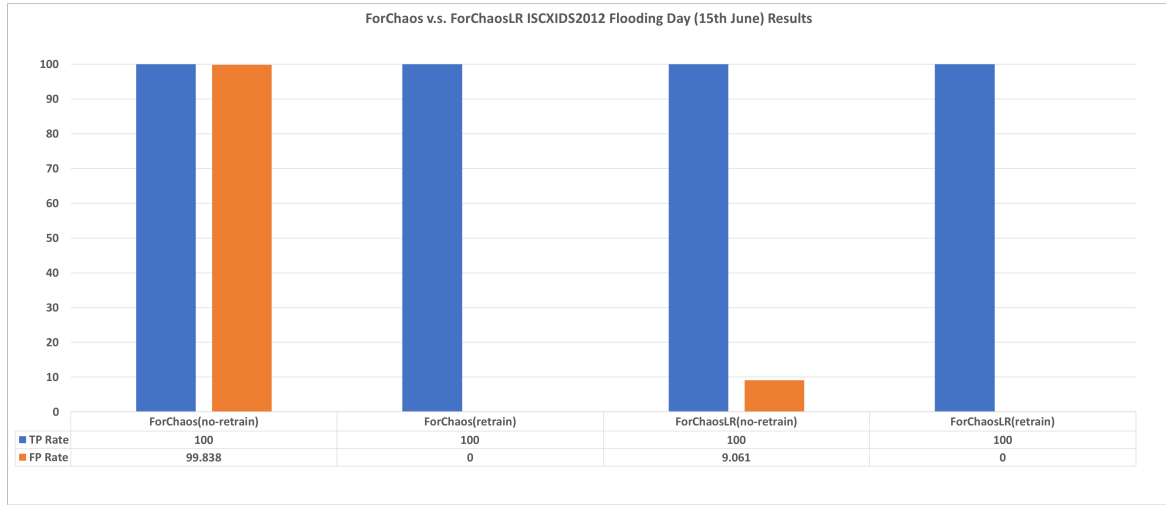


Fig. 7.4 ForChaos v.s. ForChaosLR TP and FP rates on ISCXIDS2012 Flooding Day (15th June)

conclude that the FP rate achieved by ForChaos first-layer is not acceptable. Therefore, ForChaos should be re-trained frequently and not just after a DDoS attack occurs when it is deployed in ICT networks. To evaluate our assumptions we increased the frequency of re-training of ForChaos first-layer. To find the most optimal set of parameters used (the values that could give us a high TP rate and a low FP rate), training time, α and window size w , we conducted a series of experiments.

Initially, we used the optimal parameters from the previous series of experiments in Chapter 6 Table 6.16, $\alpha = 0.1$, window size = 40 seconds and training time = 1000 seconds. However, this resulted in a very high 84.372% FP rate. After a series of different combinations of α and window size w , we found the combination of $\alpha = 0.5$ and window size = 120 seconds to be particularly effective as it reduces the FP rate to 24.701% with no re-training, an approximate 60% difference which is highly significant as illustrated in Figure 7.8. ForChaos manages to maintain the Lyapunov exponents negative for approximately 21000 seconds.

Finally, we concluded that with four re-training procedures conducted at approximately 11000 seconds, at 24100 seconds, 57300 seconds and at 60800 seconds ForChaos first-layer is able to minimise the FP rate to 18.750%. Re-training makes ForChaos first-layer is more stable and therefore the Lyapunov exponents remain negative for the majority of time with some temporary spikes since the difference is approximately 6% from the no-retrain period, as illustrated in Figure 7.5.

Such a behaviour was expected from ICT networks, due to the dynamic population of devices and hosts observed in such networks while this is less likely to occur in resource-constrained networks. Furthermore, wireless ICT networks are vastly different to wireless

resource-constrained networks as stated in [3] home-smart grid network. Some examples of resource-constrained networks include their data rate having a maximum rate of 250 kbps/sec for ZigBee networks (a well known and popular wireless protocol for resource-constrained networks) and a 11,000+ kbps for WiFi networks.

ForChaosLR multi-layer IDS was primarily designed and developed for the smart home-smart grid network and other low-resource constrained networks where the dynamic addition of devices is less likely and therefore, the traffic is less likely to go through sudden changes. Despite this, LR second-layer is able to once again minimise the FP rate to 0% regardless of whether re-training is conducted by ForChaos first-layer. However, it is good practice for ForChaos to produce as little FP rate as possible.

In summary, based on the results shown in Figure 7.9 we observe that ForChaos first-layer's FP rate is vastly decreased when re-trained by approximately 74% difference which is highly significant. We also conclude that regardless of ForChaos being re-trained or not, LR second-layer is able to minimise the FP rate significantly, giving the ForChaosLR multi-layer a final 2.318% FP rate which is a vast difference of approximately 86% when it is not re-trained and a 0% FP rate when it is re-trained with a difference of nearly 15%. Hence, we conclude that LR second-layer is very robust against FP rates, as the literature suggests [11], [107], [184], [221], [11], [107] and [184].

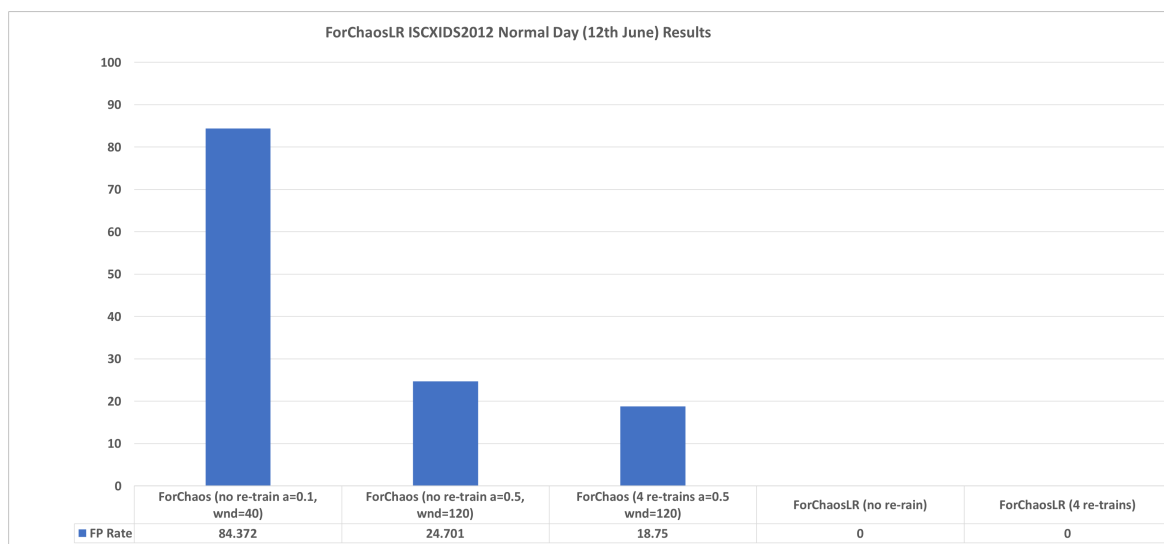


Fig. 7.5 ForChaos and ForChaosLR ISCXIDS2012 12th June Normal Behaviour Results

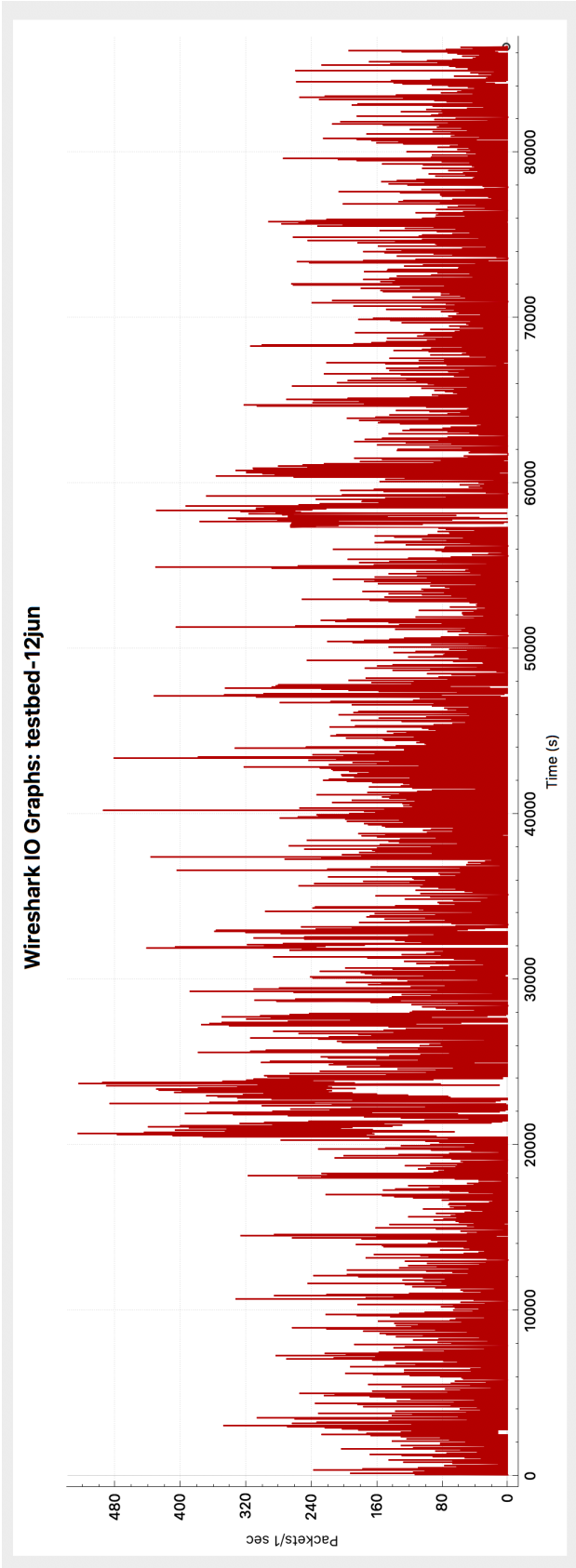


Fig. 7.6 ISCXIDS2012 12th June Normal Behaviour Packet Arrival Time/sec

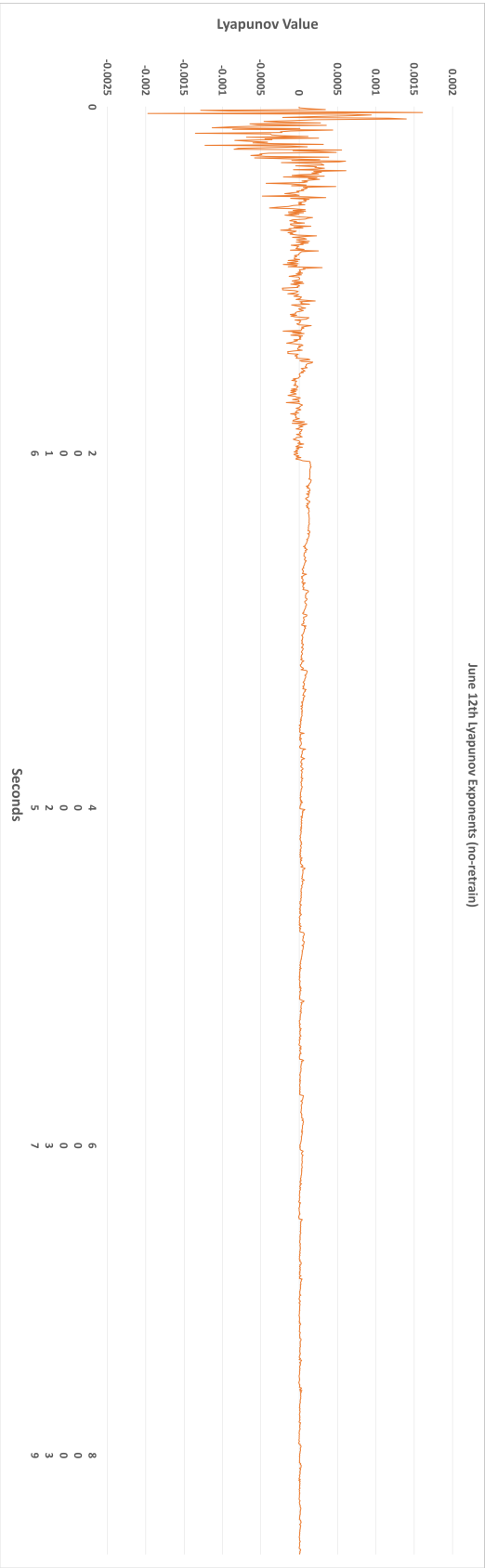


Fig. 7.7 ISCXIDS2012 12th June Normal Behaviour Lyapunov Exponents Behaviour

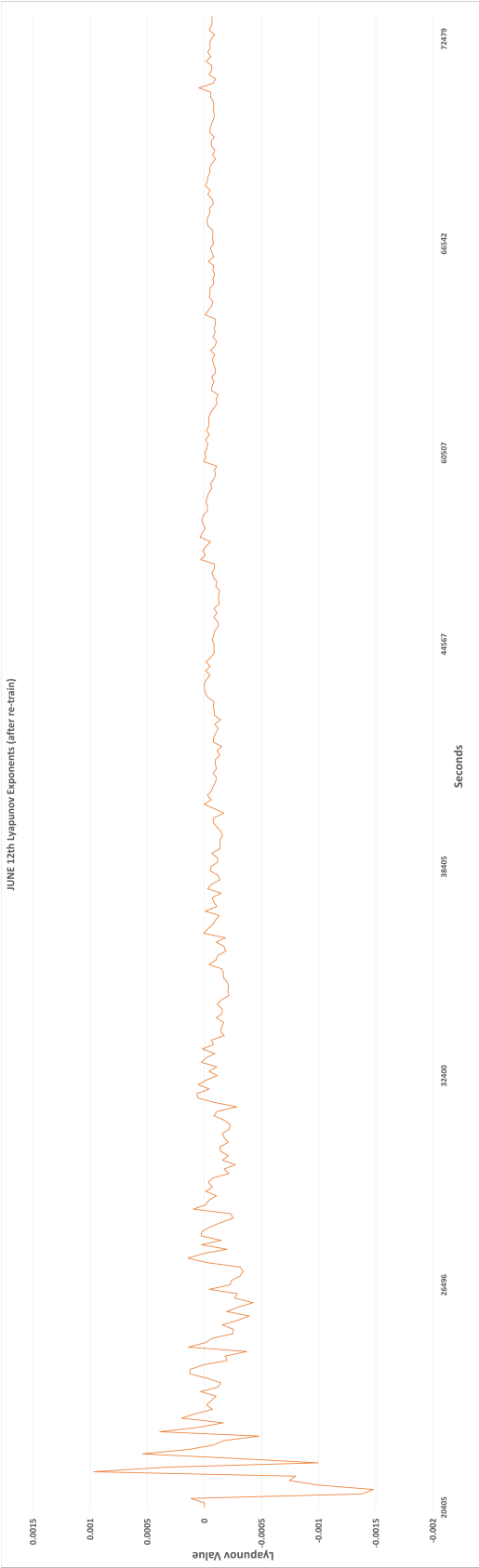


Fig. 7.8 ISCXIDS2012 12th June Normal Behaviour Lyapunov Exponents Behaviour with Re-training

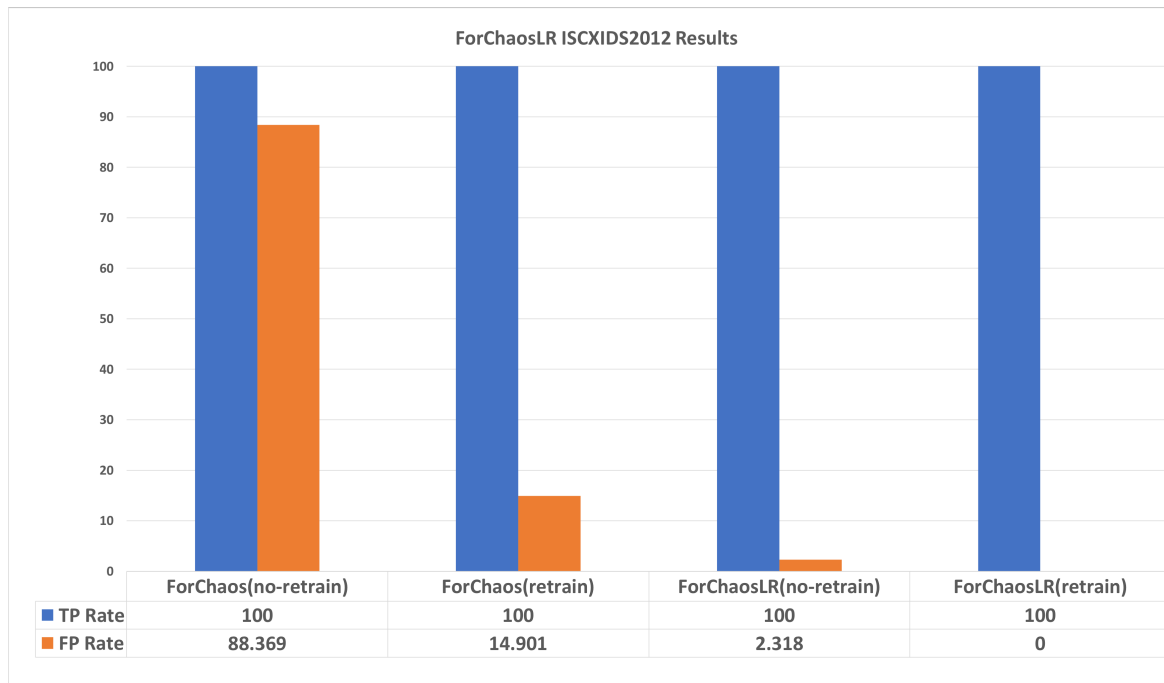


Fig. 7.9 ForChaos and ForChaosLR ISCXIDS2012 Results (total)

7.2.6 ForChaosLR compared to ML Algorithms against ISCXIDS 2012 Dataset

Overview

To evaluate the accuracy of ForChaosLR multi-layer IDS in further using other datasets, we used a number of popular machine learning algorithms, specifically: Bayesian networks (BN), naive Bayesian (NB), K-nearest neighbour (KNN), support vector machine (SVM), decision tree (DT), random forest (RF) and artificial neural networks with the multi-layer perceptron architecture (MLP).

Experimental Set Up

For the MLP, KNN and SVM algorithms the following key-parameters were tweaked in an attempt to examine whether their results can be potentially improved: learning rate for MLP, nearest neighbour along with distance and weighting used for KNN; and kernel type for SVM. The specific parameters used for every algorithm can be found in Table 6.19. Weka [144] was used for all the algorithms. Regarding training, the training set used was the same as the one used for LR second-layer. Regarding testing, the test set used was the same as the one ForChaosLR was evaluated with.

Results Analysis and Discussion

The results of the ML algorithms for the ISCXIDS2012 dataset are presented in Figures 7.10, 7.11, 7.12 and 7.13. Regarding the results for the ISCXIDS2012 dataset we drew a set of main conclusions presented below.

- All ML algorithms achieved high TP rates with the lowest being 97.849%.
- Naive Bayes, SVM(Linear), KNN (regardless of K number and distance measure used) and decision tree algorithms detected the attacks fully.
- There is an evident trend where the more "simplistic" algorithms achieve demonstrate higher accuracy than the more complex ones belonging in the same category. Some examples include naive Bayes performing better than Bayesian networks, SVM(Linear) better than SVM(Quadratic), SVM(Cubic) and SVM(RBF) and decision tree better than random forest.
- This trend does not apply to the neural network and KNN algorithms.

Based on the results, we can assume that algorithms with higher complexity and sophistication tend to underperform against a simplistic classification. We refer to the classification of the ISCXIDS2012 dataset as simplistic because no slow-rate attacks are involved and the magnitude of the flooding instances is evident, as observed from Figure 7.1. Proceeding, we compare the results gained from the algorithms that belong to the same category.

Bayesian Algorithms:

- Examining the Bayesian algorithms we observe that Bayesian network algorithm's TP rate is lower than Naive Bayes' by approximately 1.1%, a difference which is not highly significant.
- In contrast, Bayesian network's FP rate is higher than naive Bayes algorithm's with approximately 0.4%, a difference which is significant.

Neural Network Algorithms:

- Examining the neural network MLP results, we observe that both MLP models generate the same TP rate and no FP rate.
- This suggests that increasing the learning rate does not improve nor deteriorate the TP and FP rates.

SVM Algorithms :

- SVM(Linear) and SVM(RBF) generate the highest TP rates with a full accuracy rate, followed by SVM(Quadratic) and SVM(Cubic). The difference in TP rates are approximately 0.5% and 1.5% respectively, which are not highly significant.
- SVM(RBF) showed a considerable FP rate with 6.727% which is approximately 5% higher than SVM(Cubic)'s FP rate and 6.5% than SVM(Quadratic) and SVM(Linear).
- Based on these results, we believe that SVM(RBF) is not accurate enough in classifying normal traffic as legitimate as polynomial kernels are. In addition, the higher the degree of a polynomial kernel, the higher the generated FP rate is.

KNN Algorithms:

- All the KNN algorithms regardless of the distance measure or the K number achieved a 100% TP rate .
- Regarding FP rates, we observed mixed results. In detail, all of the combinations of KNN produce a FP rate between 4.375-25.506% with the exception of KNN with Manhattan distance used.
- This shows that the KNN algorithm presents issues in correctly classifying normal behaviour as legitimate as opposed to the other ML algorithms used. In more detail:
 - Manhattan distance is robust against FP rates as it does not misclassify any normal instances as malicious.
 - In contrast, Chebyshev produces the worst results with 25.506% FP rate, regardless of the K number used.
 - Euclidean and filtered Euclidean distance produce similar results with the latter producing the lowest FP rate, 3.962%, when K=3 is used. This was expected as filtered Euclidean is an enhanced version of Euclidean distance which was intended to improve overall results.
 - Furthermore, when Euclidean distance is used, the FP rate is decreased as the K number is increased.

Tree Algorithms:

- Decision tree algorithm's TP rate is 100% while the TP rate is 98.925%, an approximate difference of 1.075%, which is not highly significant.

- Regarding FP rates we observe that both do not generate any FP rates.
- Hence, we conclude that tree-based algorithms are highly accurate in differentiating between flooding and legitimate traffic.

ForChaosLR results and comparison with ML algorithms:

- ForChaosLR's TP rate remains unaffected with re-training. This indicates that ForChaos first-layer is able to fully detect the flooding DDoS attack without any re-training.
- The FP rate is decreased from 2.318% to 0% after re-training. Based on the results from Figure 7.9 the FP rate generated from ForChaos first-layer is significantly high (88.369%) and moderate (14.901%) when training is considered or not respectively.
- LR second-layer manages to reduce the overall FP rate to 2.318% and 0% respectively. These findings are supported by the literature presented in Chapter 3 where logistic regression was proved particularly effective in reducing FP rates [11], [107], [184] and [216].
- Comparing the results with ForChaosLR multi-layer IDS we can observe that our proposed IDS outperforms sophisticated algorithms such as MLP, Random Forests, SVM (RBF) and Bayesian Networks in TP rates as it achieved a full 100% TP rate regardless of ForChaos first-layer being retrained or not.
- Regarding FP rates, ForChaos multi-layer IDS produces a 2.318% FP rate when ForChaos first-layer is not retrained surpassing all KNN algorithms, with the exception of the KNN algorithms which Manhattan distance is used. However, if ForChaos is re-trained after flooding attack finishes the FP rate is reduced to 0% as the majority of FP rate produced by ForChaos multi-layer IDS was from the traffic falsely forwarded to LR second-layer after the flooding attack was conducted.
- We should also point out that the ISCXIDS 2012 only contains flooding and legitimate traffic. Hence, regardless of the algorithmic model used, the evaluation results are likely to be more accurate as the problem to be solved (flooding v.s. legitimate traffic) is less complex.

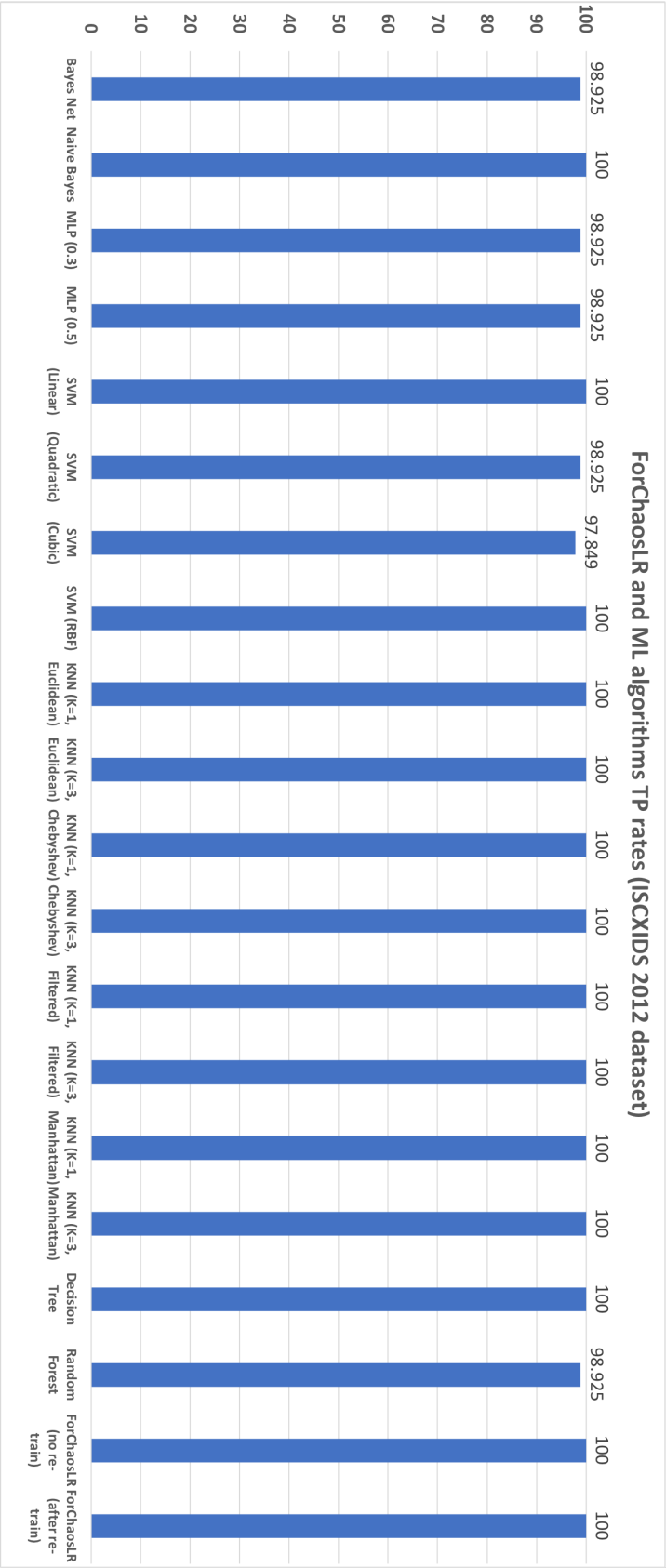


Fig. 7.10 ISCXIDS2012 ForChaosLR and ML algorithms TP rates

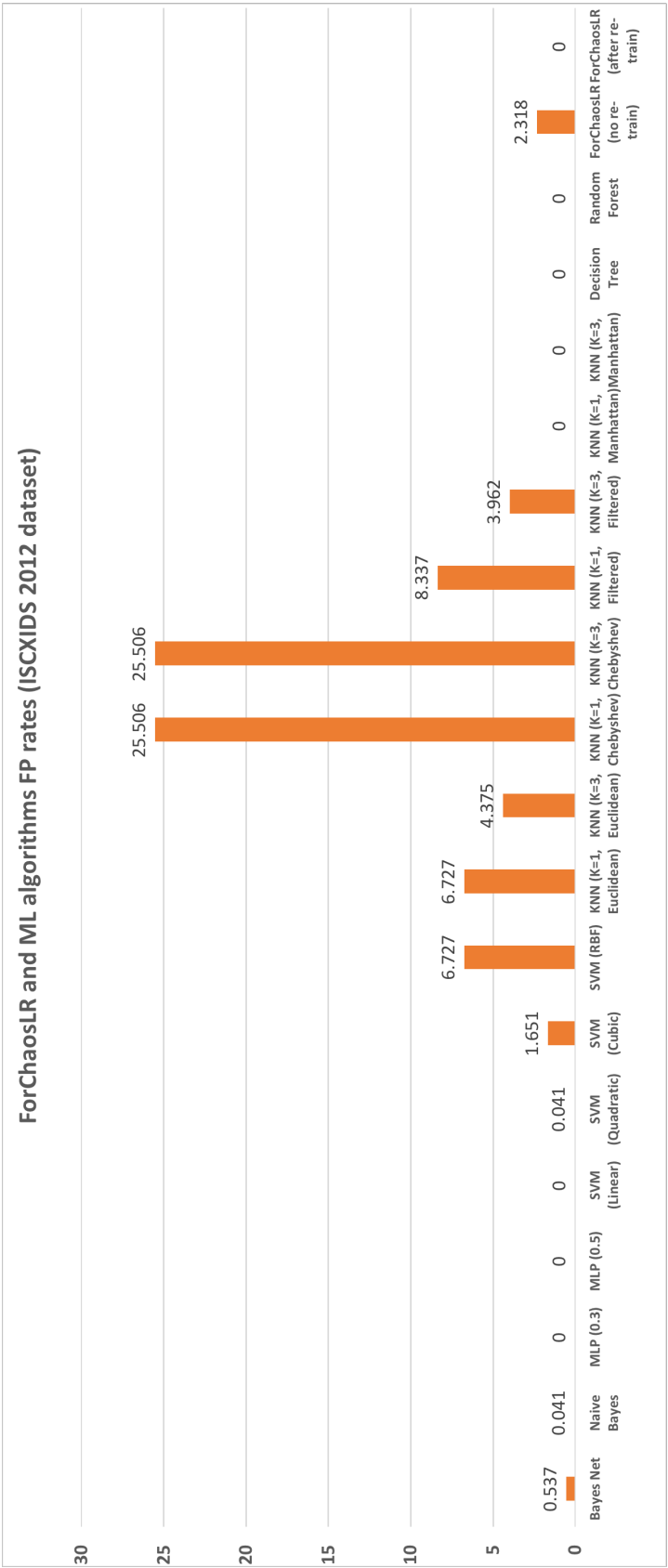


Fig. 7.11 ISCXIDS2012 ForChaosLR and ML algorithms FP rates

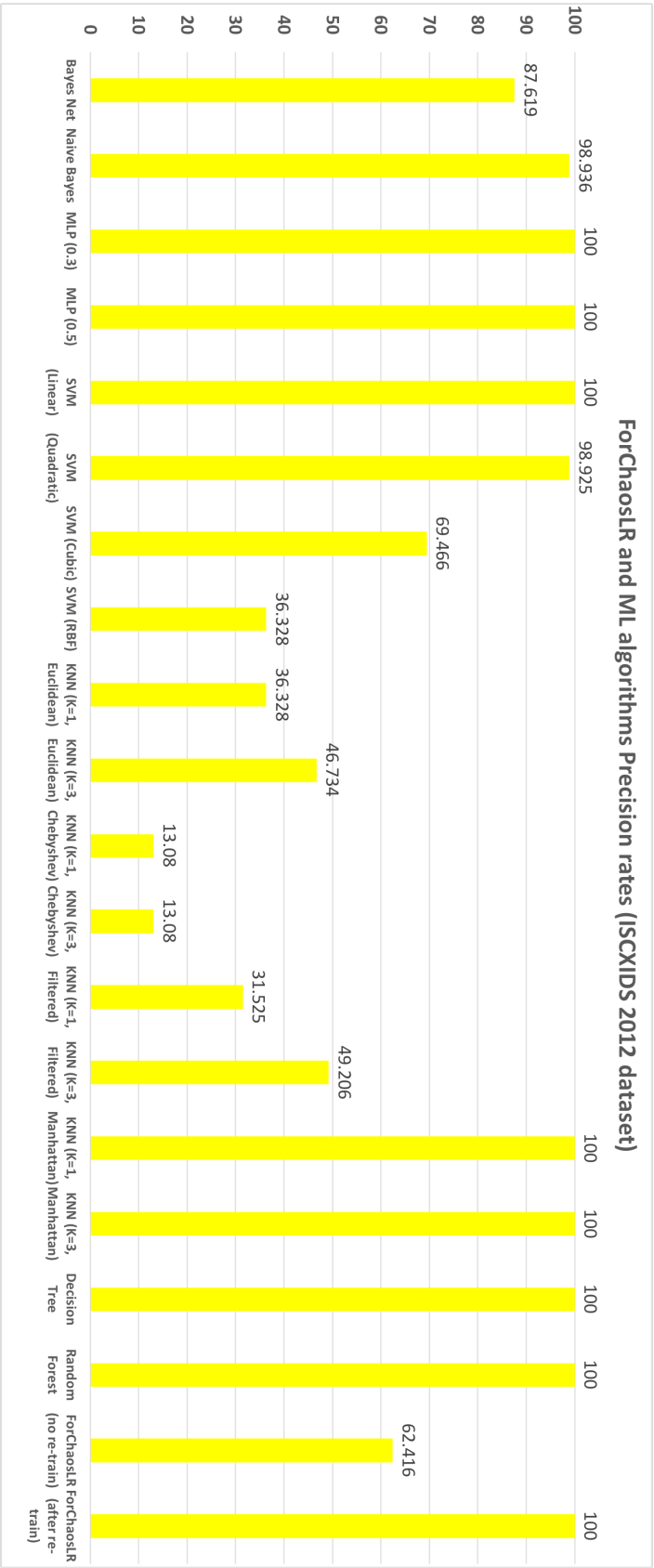


Fig. 7.12 ISCXIDS2012 ForChaosLR and ML algorithms Precision rates

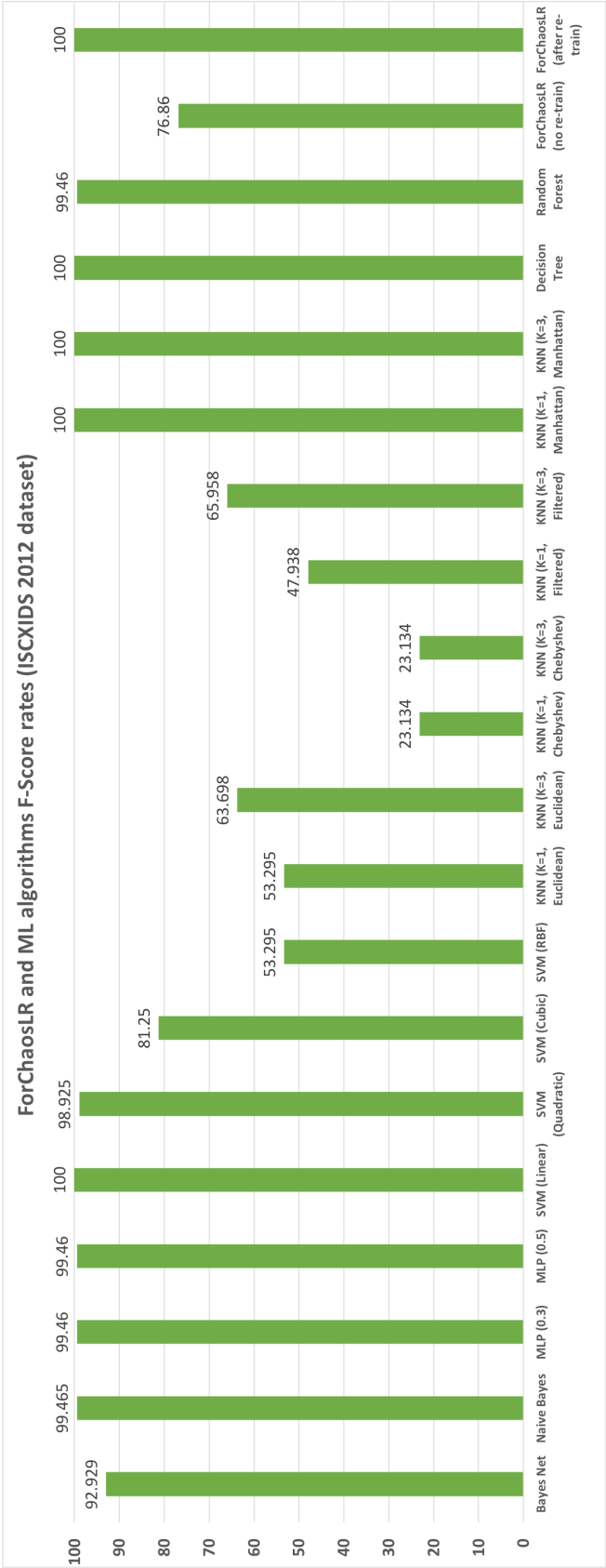


Fig. 7.13 ISCXIDS2012 ForChaosLR and ML algorithms F-Score rates

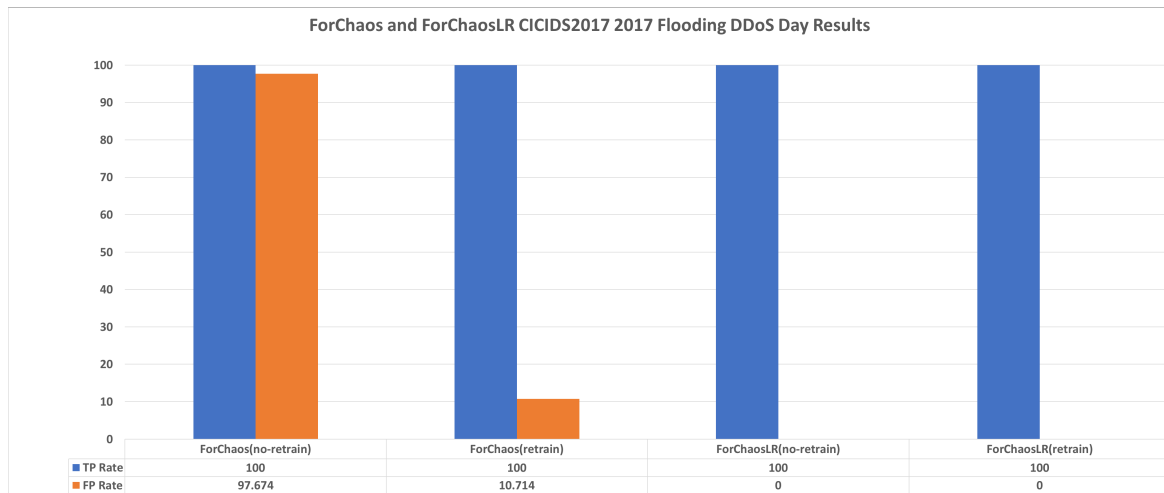


Fig. 7.14 ForChaos and ForChaosLR 2017 Results Flooding Day TP and FP rates

7.2.7 Evaluation of ForChaosLR against CICIDS 2017 Dataset

- From the results in Figure 7.14 we observe that ForChaosLR multi-layer IDS is able to fully detect the flooding attack in all of its duration. However, we once again notice that ForChaos first-layer when not re-trained achieves a very high FP rate of 97.674%.
- By examining the results in detail, we once again note that both ForChaos first-layer and LR second-layer are able to fully detect flooding attacks as it was previously observed in Figure 7.4. This is justified due to a flooding attack's magnitude and impact it has on network traffic.
- However, once again despite ForChaos first-layer's accuracy, it achieves a high FP rate, specifically 97.674 %, after the attack is conducted for the same reasons described in the previous section. We retrained ForChaos' first-layer after the attack finishes using the following parameters: 1000 seconds of training time, $\alpha = 0.1$ and a window size = 40 seconds. This set of parameters was proved sufficient enough as ForChaos first-layer achieves a 10.714% FP rate, an approximate 87% difference which is highly significant. Regardless of ForChaos first-layer being re-trained, LR second-layer once again manages to reduce the FP rate to 0% which proves its robustness and effectiveness in correctly classifying legitimate traffic.
- From the results in Figure 7.15 we observe that in contrast to previous experiments, ForChaosLR multi-layer IDS's TP rate accuracy falls by 3%, as ForChaos first-layer is able to fully detect the slow-rate attacks but LR second-layer is not. This was expected,

as slow-rate attacks are stealthier and achieve less "noise" than flooding so it is harder to detect.

- One of our proposed system's objectives is to detect slow-rate DDoS attacks. This objective is satisfied, as ForChaosLR multi-layer IDS fully detects this type of attack. Furthermore, ForChaosLR multi-layer IDS detects Slow HTTP Test and HULK DDoS attacks as shown in Table 7.3. For ForChaosLR multi-layer IDS to be able to detect Goldeneye DDoS attacks more investigation is needed that possibly involves an extension in features monitored.
- Once again, based on the results in Figure 7.15, ForChaos first-layer achieves a high FP rate, specifically 93.916%. This is slightly lower rate from the FP rate regarding the flooding attack (97.674 %) but still very high regardless. Hence, we retrained ForChaos first-layer after the attacks finish. For initial parameters, we used the optimal parameters found previously, $\alpha = 0.1$, window size = 40 seconds and training time = 1000 seconds with one re-train phase when the attacks finish at approximately 9500th second. However, this resulted in a still very high 77.470% FP rate.
- Therefore, we changed the parameters to another set found previously to be successful ($\alpha = 0.5$, window size = 120 seconds and training time = 1000 seconds) with one re-train phase when the attacks finish at approximately 9500th second. This resulted in a much lower FP rate of 43.137%. To improve our results further, we introduced two additional phases of re-training at approximately the 17100th and 20900th second. The triple re-training reduced the FP rate further to 27.957%.
- LR second-layer is able to reduce the FP rate to 6.084% when ForChaos is not re-trained in any way and to 5.263% when double re-training is introduced to ForChaos first-layer during the slow-rate DDoS day. This once again shows LR second-layer's robustness and effectiveness in correctly classifying legitimate traffic.
- In summary, we present ForChaosLR's results for the total of CICIDS 2017 dataset in Figure 7.16.
- We observe that LR second-layer is able to reduce the FP rate achieved by ForChaos (when it is not re-trained) by approximately 70%. This proves LR's ability in reducing high FP rates as the literature, presented in Chapter 2, Section 3.5 suggests.
- We observe that ForChaos first-layer's FP rate is significantly decreased when re-trained by approximately 95% (when ForChaos first-layer is not retrained), which is

Table 7.3 ForChaosLR multi-layer CICIDS 2017 Application Layer DDoS Day Individual Results

	Slowloris Rate	Slow HTTP Test Rate	HULK	Goldeneye	Total
ForChaos first-layer	100	100	100	100	100
LR second-layer	100	100	100	76.923	97

a highly significant difference. We also note that ForChaos first-layer is particularly highly accurate in slow-rate attacks as it is able to detect them regardless of their type since the TP rate is 100%.

- We also conclude that although LR second-layer reduces the TP rate by approximately 2.3% it is able to minimise the FP rate greatly, giving the ForChaosLR multi-layer a final 4.53% FP rate which is a vast difference of approximately 90% when ForChaos first-layer is not re-trained. Hence, we conclude that LR second-layer is very robust against FP rates.
- These results directly agree with the literature's findings, where a logistic regression model is effective in reducing FP rates as the studies in [11], [107], [184], [221], [11], [107] and [184] supported.

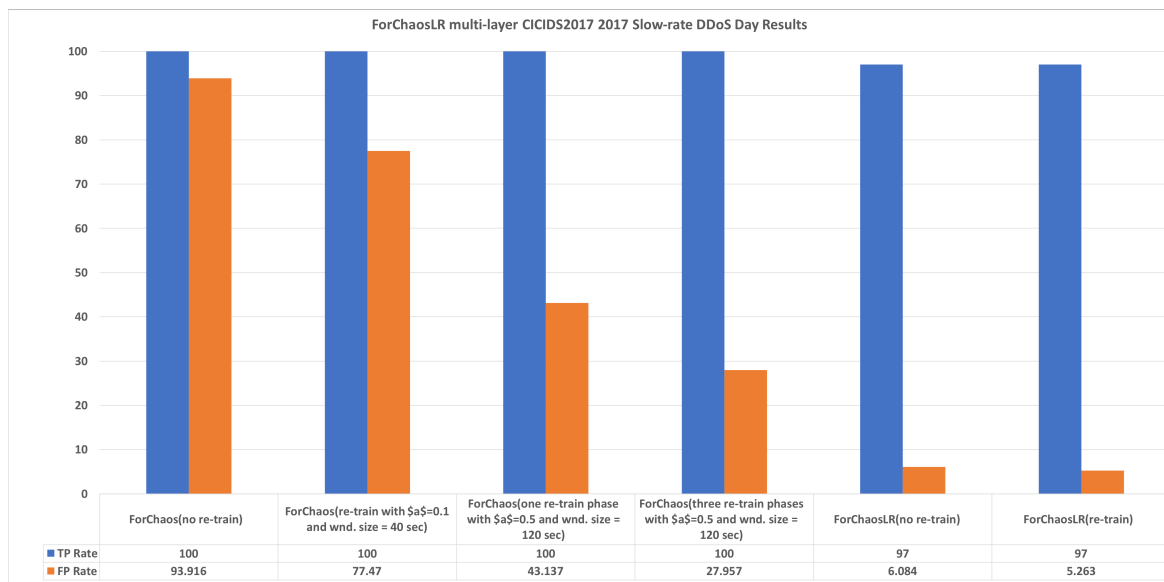


Fig. 7.15 ForChaos and ForChaosLR 2017 Results Slow-Rate Day TP and FP rates

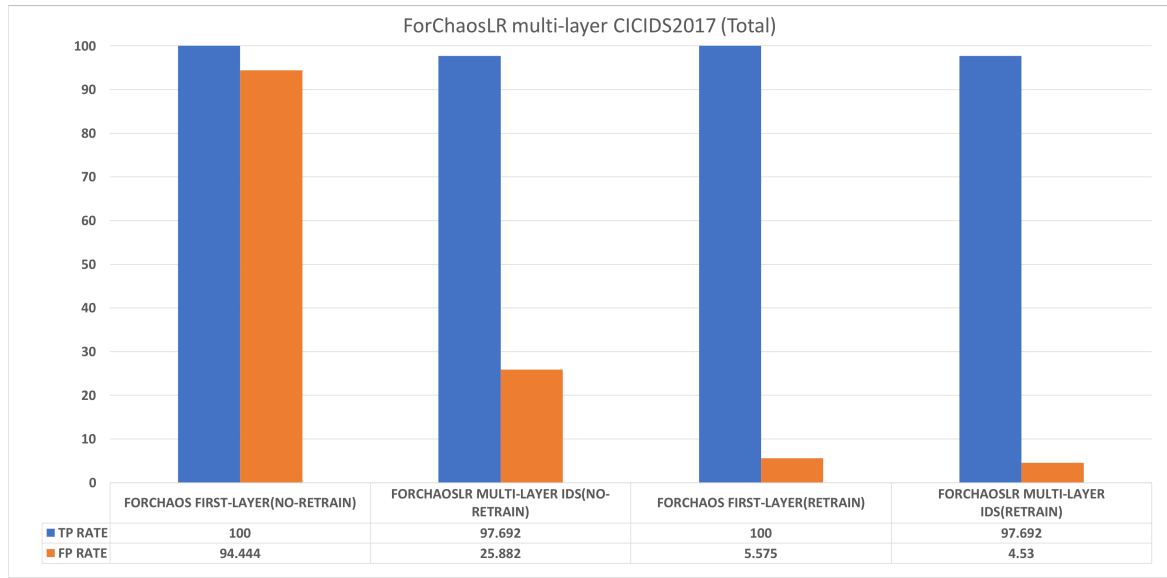


Fig. 7.16 ForChaos and ForChaosLR CICIDS 2017 Results TP and FP rates (total)

7.2.8 ForChaosLR compared to ML Algorithms against CICIDS 2017 Dataset

Motivation

To evaluate the accuracy of ForChaosLR multi-layer IDS with other datasets, we used the same set of machine learning algorithms as in Chapter 6. More information regarding the parameters used in the construction of the models can be found in Table 6.19.

Experimental Set Up

For the MLP, KNN and SVM algorithms, the following key parameters were investigated to improve results: learning rate for MLP, nearest neighbour along with distance and weighting used for KNN and kernel type for SVM. The specific parameters used for every algorithm can be found in more detail in Table 6.19.

Weka library [144] was used for the implementation of all the algorithms. We used the same training set as the one used for LR second-layer. Likewise, we used the same test set as the one for evaluating ForChaosLR.

Results Analysis and Discussion

The results of the ML algorithms for the CICIDS 2017 are presented in Figure 7.17, 7.19, 7.19 and 7.20. Based on the results, we draw the following conclusions:

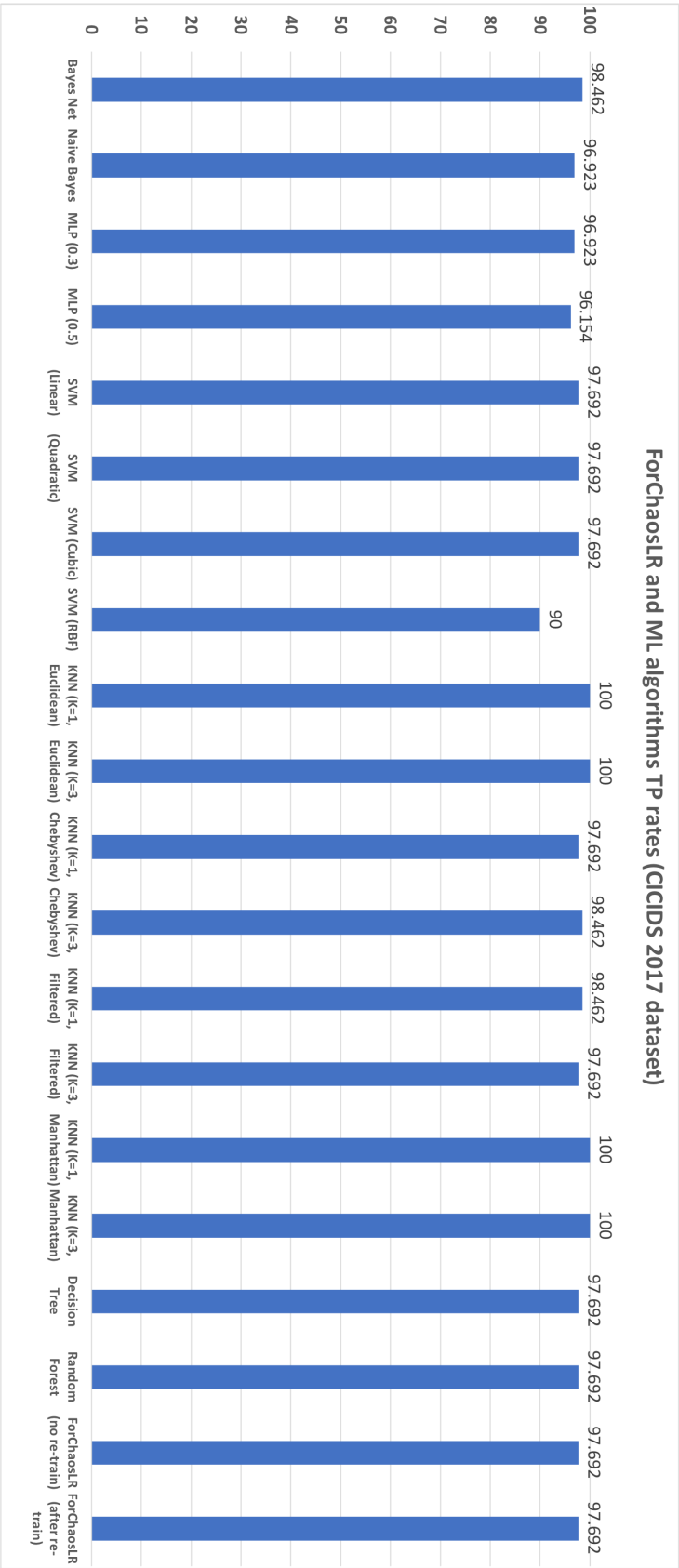


Fig. 7.17 CICIDS 2017 ForChaosLR and ML algorithms TP rates

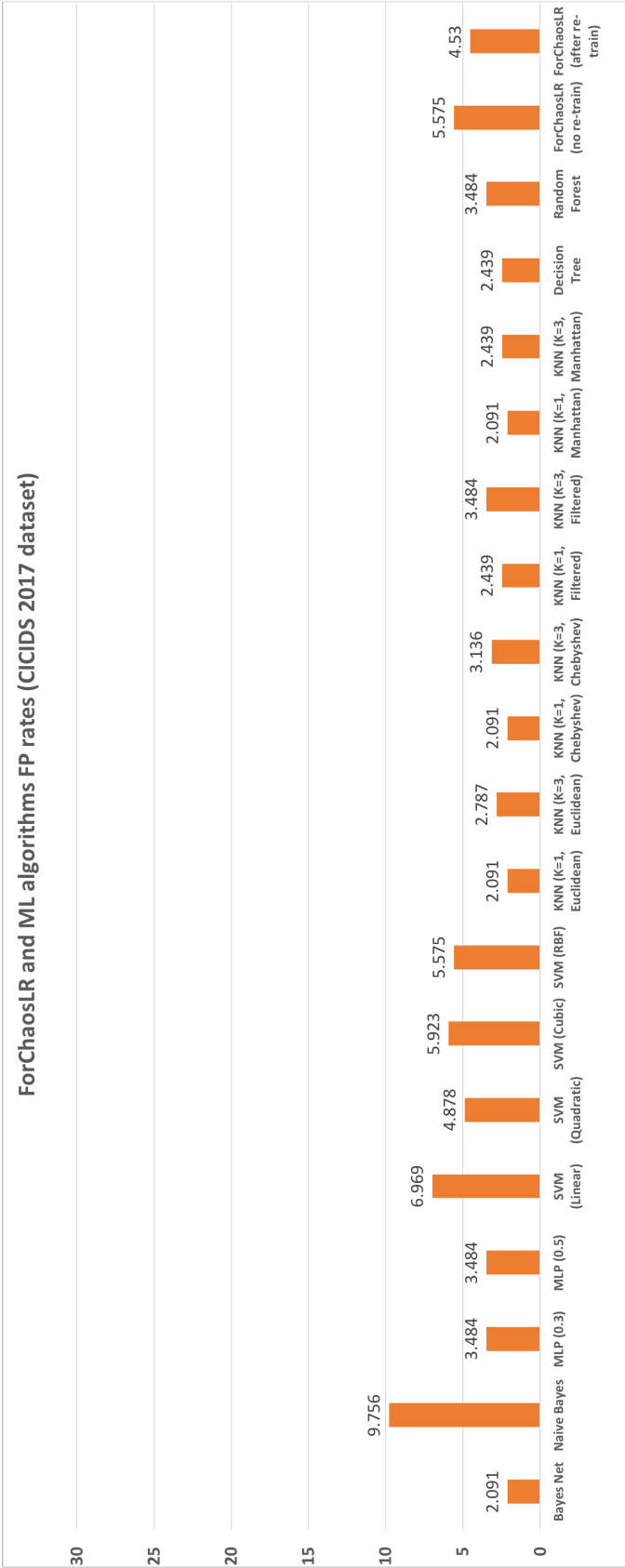


Fig. 7.18 CICIDS 2017 ForChaosLR and ML algorithms FP rates

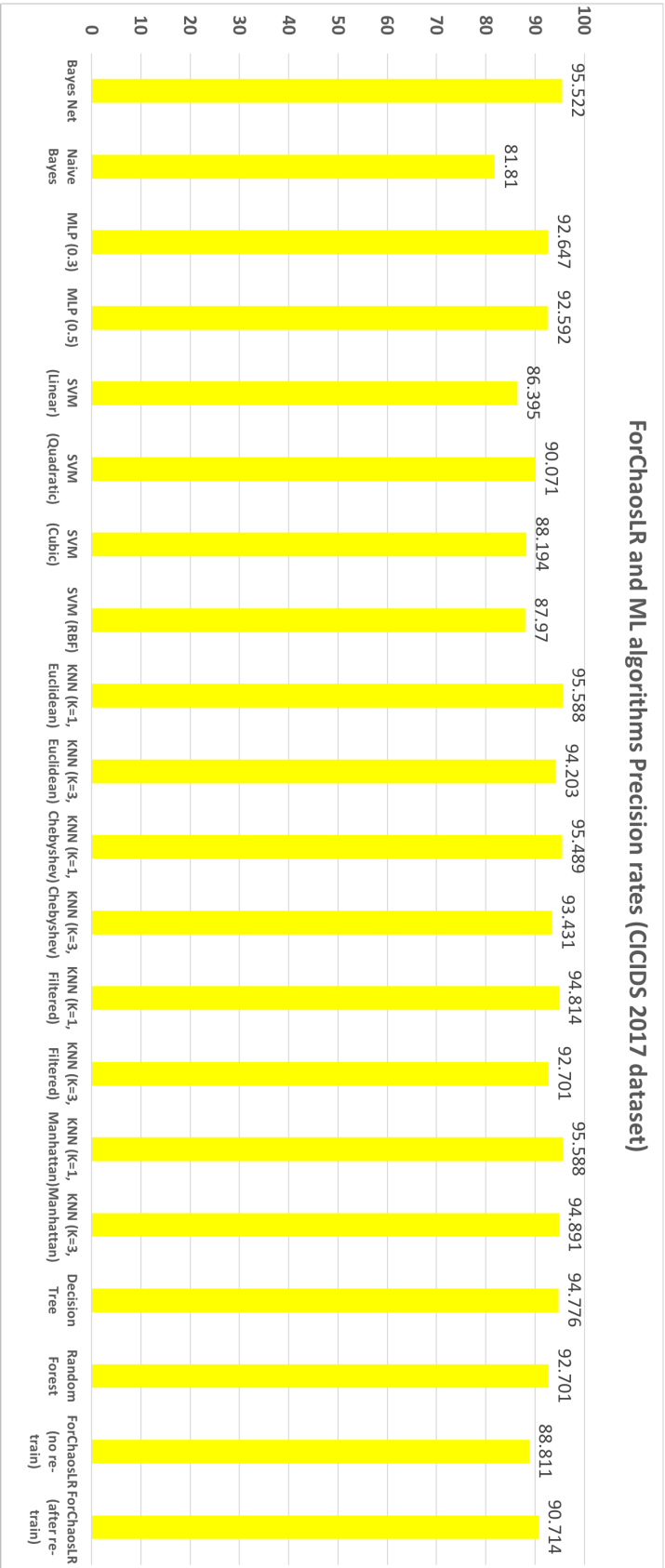


Fig. 7.19 CICIDS 2017 ForChaosLR and ML algorithms Precision rates

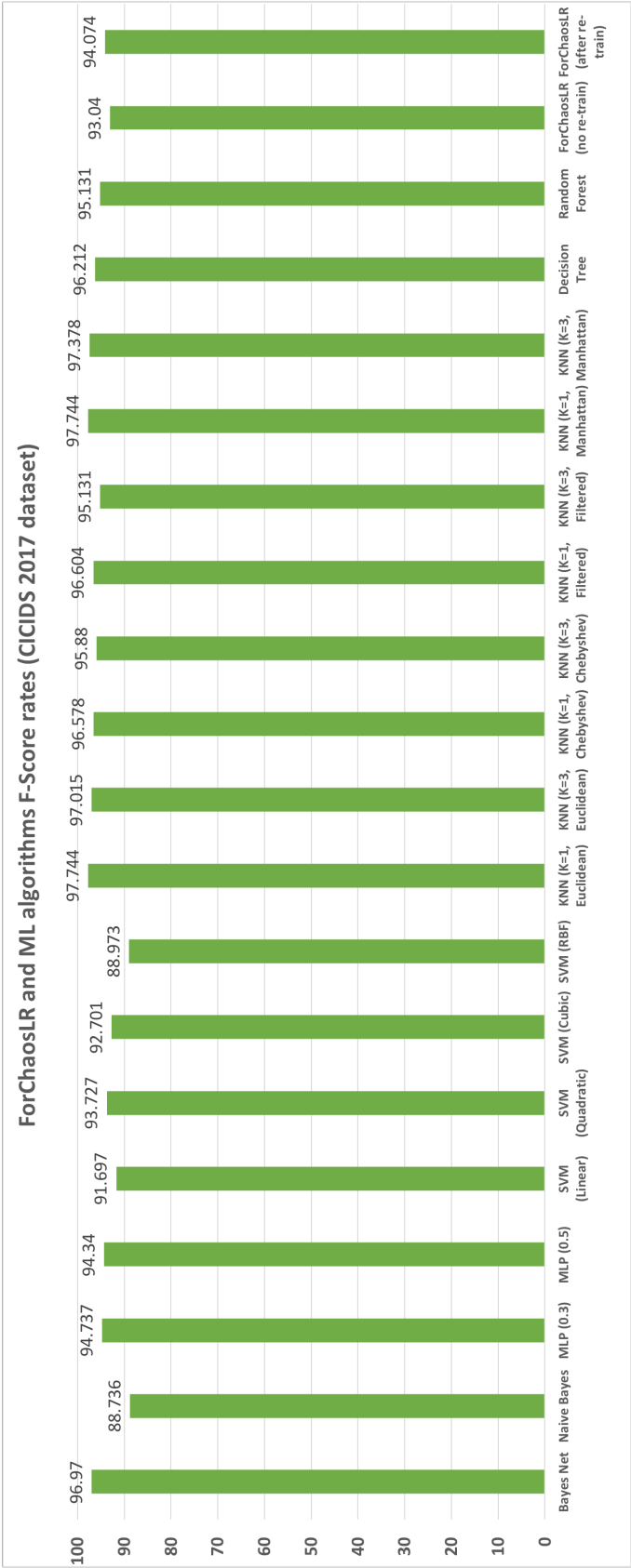


Fig. 7.20 CICIDS 2017 ForChaosLR and ML algorithms F-Score rates

- All ML algorithms achieved high TP rates, with the lowest being 96.154%.
- Based on their TP rates, the ML algorithms chosen are robust against slow-rate attacks despite their stealthy nature.
- The majority of ML algorithms performed worse than the results generated in ISCXIDS 2012 dataset both in TP and FP rates. This was expected as application layer slow-rate DDoS attacks are stealthier than flooding attacks.
- Only KNN using Manhattan and Euclidean distance measures managed to fully detect the slow-rate attacks.
- KNN (Manhattan) and KNN(Euclidean) results' robustness was also evident in the ISCXIDS 2012 dataset results.
- With the exception of KNN Chebyshev and Filtered distances, the rest of the ML algorithms produced higher FP rates than their ISCXIDS 2012 respective FP results.
- Furthermore, slow-rate attacks are much stealthier and do not generate a lot of "noise" in traffic as opposed to flooding attacks and therefore they are much similar to legitimate traffic.
- In contrast to the results from the ISCXIDS 2012 dataset, the more complex algorithms from each category perform better than their respective simplistic ones.
- This trend was expected since multiple attacks are to be detected which can resemble legitimate traffic to a great degree.

Proceeding, we compare the results gained from the algorithms that belong to the same category.

Bayesian Algorithms :

- In contrast to the results gained from ISCXIDS 2012 dataset, the more complex Bayesian networks perform more accurately than their respective simplistic naive Bayesian.
- This is due to slow-rate attacks being stealthier and therefore harder to be detected than flooding attacks. In more detail, there is an approximate 1.5% difference in TP rates between them.

- Regarding FP rates, Naive Bayes produces a high FP rate with nearly 8% difference from Bayesian Networks. This was expected, as Naive Bayes is a simplistic algorithm as opposed to Bayesian Networks which consists of a complex probabilistic network.

Neural Network Algorithms :

- Examining the MLP algorithms, we observe that MLP(0.3) produces a slightly higher TP rate than the MLP(0.5) with an approximate difference of 0.8% which is not highly significant.
- In contrast, the FP rate does not change so the learning rate does not affect the FP rates but it does affect the TP rate.

SVM Algorithms :

- SVM with polynomial kernel results are clearly more accurate in TP rates than SVM(RBF) which means that a polynomial kernel has a clear advantage over the RBF kernel with approximately 7.5%.
- However, SVM(Linear) achieves the highest FP rate, followed by SVM(Cubic), SVM(RBF) and SVM(Quadratic). However the differences are not highly significant as it is approximately less than 2% from the highest to the lowest.
- Based on results, more complex SVM kernels achieve less FP rates than the more simplistic ones. SVM(Linear) produces the highest FP rate with 6.873%, making it the algorithm with the worst performance while SVM(Quadratic) has the best.
- Although FP rate is increased when cubic and RBF kernels are used, it is still lower than the FP rate produced by the SVM(Linear).
- This was expected, as the current classification problem is unlikely to be effectively solved linearly due to the nature of the flooding and slow-rate attacks as well as the slow-rate attacks' similar behaviour to normal traffic.

KNN Algorithms :

- There is a noticeable drop in TP rates as opposed to their perspective results for the ISCXIDS 2012 dataset but by only a maximum of 2.308%.
- Manhattan and Euclidean distances proved to be very effective in detecting both flooding and slow-rate attacks achieving a 100% TP rate.

- The K number does not make any difference regarding the Manhattan and Euclidean distance measures. However, K number improves TP rates for the remaining two distance measures, Chebyshev and filtered.
- The TP rates are improved by 0.77 in Chebyshev distance measures when K number is set to 3. This was expected as a higher number of K denotes more neighbour instances being involved in classification.
- This makes a positive impact as it is more complex to classify between flooding, slow-rate and normal traffic. Interestingly that was not the case for filtered KNN when the K number is increased as the TP was decreased from 98.462% to 97.692% TP rate. However, the 1% difference cannot be considered significant.
- Regarding FP rates, when K number is increased the FP rate increases regardless of the distance measure used. This once again highlights the issues of constructing an algorithmic model to detect both flooding and slow-rate attacks.

Tree Algorithms :

- Regarding tree-based algorithms we see no difference between decision tree and random forest algorithms. This indicates that dividing the training set and training multiple decision trees does not make any difference in TP rates.
- Regarding FP rates, decision tree algorithm achieves a lower FP rate than random forests by approximately 1% difference.
- Although both of their FP rates are not very high, this difference does reflect that the division of the training set and construction of different trees does not seem particularly effective in terms of correctly classifying incoming traffic as legitimate in this dataset.
- We suspect that an increase in training set is likely to improve results.

ForChaosLR results and comparison with ML algorithms:

- Similarly to the majority of the ML algorithms, ForChaosLR achieves a lower TP rate and a higher FP rate compared to the results from the ISCXIDS 2012 dataset. This was expected since in the previous dataset only flooding DDoS attacks were present so it was easier to make the distinction between flooding and legitimate traffic.
- ForChaosLR multi-layer IDS is able to maintain a very high TP rate of 97.692%, higher than naive Bayesian algorithm and both neural network (MLP) algorithms (0.3 and 0.5 learning rates) and SVM(RBF).

- ForChaosLR achieves the same TP rate as tree algorithms (decision tree and random forest algorithms), all polynomial kernel SVM algorithms, KNN with Chebyshev distance and $K=1$ and KNN with filtered distance and $K=3$.
- When compared to the ML algorithms that achieved a higher TP rate than ForChaosLR (Bayesian networks, KNN (Euclidean and Manhattan), KNN(Chebyshev $K=3$) and KNN(Filtered $K=1$)), we observe that the difference is of a maximum of approximately less than 2.5%,
- ForChaosLR when re-trained is able to achieve a FP rate of 4.530% which is lower than naive Bayesian and all SVM algorithms.
- Although ForChaosLR achieves a higher FP rate than the rest of the ML algorithms, the difference is at a maximum of approximately 2.4%.

7.3 Evaluation of ForChaosLR using Smart Home Dataset

7.3.1 Motivation

Although there is no real-world dataset consisting of smart home-smart grid traffic combined with application layer DDoS attacks we evaluated ForChaosLR multi-layer IDS against legitimate behaviour in a real-world smart home network using the dataset provided from [206] and [4] and presented in Chapter 5, Section 5.2.2. Proceeding, after conducting experiments using the NS-3 dataset generated in our smart home-smart grid network simulation we proceed with evaluating our proposed IDS using real-world smart home data.

7.3.2 Experimental Setup Details

As already stated, we used two datasets, namely the 2018 and 2019 smart home datasets, Chapter 5 Section 5.2.2, both containing normal legitimate traffic. The 2018 smart home dataset consists of 20 days of normal traffic and the 2019 smart home dataset consists of 35 days of normal traffic. To construct the logistic regression model for LR second-layer we also need malicious traffic apart from legitimate traffic. Thus, we used application layer DDoS traffic from the dataset from [170], presented in Chapter 5, Section 5.2.2. This application layer DDoS traffic (Goldeneye) was from an IoT network, but not a smart home network.

Although both the 2018 and 2019 smart home datasets were generated and collected in the same smart home network, we chose to separately evaluate them. The details of the training sets constructed for the LR second-layer are stated below. For the 2018 dataset, to

Table 7.4 ForChaosLR multi-layer Smart Home 2018 Results

	<i>FP Rate (%)</i>
ForChaos first-layer	17.81
LR second-layer	0.4

evaluate LR second-layer we used 7 days for training out of 20 (and use the rest for testing) for constructing the normal instances. The attack instances constructed using the [170] dataset were increased using the SMOTE algorithm from 53 instances to 6784 instances. The total number of normal instances was 5852. For the 2019 dataset, to evaluate LR second-layer we used 7 days for training out of 35 (and use the rest for testing) for constructing the normal instances. The attack instances constructed using the [170] dataset were increased using the SMOTE algorithm from 53 instances to 3392 instances. The total number of normal instances was 6280.

7.3.3 Results Analysis and Discussion

The results of ForChaos first-layer are listed in Table 7.4 for the 2018 dataset and in Table 7.5 for the 2019 dataset. We observe that ForChaos first-layer is able to classify the majority of incoming traffic as legitimate correctly, with the FP rates being 17.81% for the 2018 dataset and 10.77% for the 2019 dataset. In comparison to the 7.054% FP rate for the NS-3 binary dataset, the difference is approximately ranging from 3-11%, which is an acceptable range of error.

Once again, LR second-layer was shown to be quite effective in reducing the FP rate from 17.81% to 0.4% for the 2018 smart home dataset and from 10.77% to 0.2% for the 2019 smart home dataset. The results agree with our NS-3 binary dataset results, where LR second-layer managed to reduce the FP rate from 7.054% to 0.52%. In addition, this set of results also confirms our literature's findings which suggest that a logistic regression model is able to significantly reduce FP rates [11], [107], [184], [221], [11], [107] and [184].

Concluding, we can argue that ForChaosLR multi-layer IDS's results from the NS-3 dataset are accurate and reflective enough since the FP rates gained from the NS-3 binary dataset and the 2018 and 2019 smart home datasets are similar. In addition, since the FP rates are similar, we can also argue that our smart home-smart grid network simulation is accurate enough with a real-world smart home traffic.

Table 7.5 ForChaosLR multi-layer Smart Home 2019 Results

	<i>FP Rate (%)</i>
ForChaos first-layer	10.77
LR second-layer	0.2

7.4 Summary

In this chapter, we presented all of the experimental results for ForChaosLR multi-layer IDS using the real-world binary datasets presented in Chapter 5, specifically ISCXIDS 2012 , Chapter 5, Section 5.2.2, CICIDS 2017, Chapter 5, Section 5.2.2 and the smart home datasets Chapter 5, Section 5.2.2. Appropriate testing evaluation were followed for both ForChaos and LR layers through different datasets. ForChaosLR is able to detect both real-world flooding and slow-rate DDoS attacks. The NS-3 dataset can be considered similar and realistic enough to real world traffic.

These validations were evaluated through the ISCXIDS 2012 and CICIDS 2017 datasets. In addition, ForChaosLR was evaluated through the two smart home datasets presented in Chapter 5, Section 5.2.2. We used these datasets so we could empirically evaluate that ForChaosLR results achieved from the NS-3 dataset are accurate enough and the synthetic traffic was similar to a real-world smart home traffic.

Furthermore, different ML algorithms were evaluated using the the different datasets so their results could be compared with the results generated from ForChaosLR multi-layer IDS. In addition, we introduced a re-training procedure for ForChaos first-layer to reduce the high FP rate. Results showed that this procedure was successful. We also conclude that ForChaosLR multi-layer IDS is able to detect other types of slow-rate attacks such and SlowHTTPTest as well as HULK and Goldeneye flooding DDoS attacks [92].

Based on the evaluation results, we observe that ForChaosLR achieves a full accuracy rate for the ISCXIDS 2012. The main reason, for this high accuracy rate, is the high magnitude of the flooding attack observed in the dataset. For the CICIDS 2017 dataset, ForChaos achieves a 97.692% TP rate, a result higher than the equivalent ones achieved by MLP neural networks and SVM(RBF) and naive Bayesian. On the other hand, ForChaosLR achieves the same TP rate as tree algorithms (decision tree and random forest algorithms), all polynomial kernel SVM algorithms, KNN with Chebyshev distance and K=1 and KNN with filtered distance and K=3. Finally, when compared to the ML algorithms which achieved a higher TP rate than ForChaosLR (Bayesian networks, KNN (Euclidean and Manhattan), KNN(Chebyshev K=3) and KNN(Filtered K=1)), we observe that the difference is of a maximum of approximately less than 2.5%,

With regards to the FP rates, ForChaosLR (when re-trained) achieves a 4.53% FP rate, lower than MLP neural networks, SVM algorithms and naive Bayesian. Although ForChaosLR achieves a higher FP rate than the rest of the ML algorithms, the difference is at a maximum of approximately 2.4%.

Based on the results summarised, we make the following conclusions:

- ForChaosLR can detect variations of flooding and slow-rate DDoS attacks such as Slow HTTP test, HULK and Goldeneye DDoS attacks.
- When re-trained, ForChaosLR achieves a lower FP rate. Specifically, for the ISCXIDS 2012 dataset the FP rate is decreased from 2.318% to 0%. For the CICIDS 2017 dataset, the FP rate is decreased from 5.575% to 4.53%. Hence, our assumption (supported by relevant studies [225], [121], [16]) on ForChaos first-layer improving its accuracy when re-trained on highly varied traffic (such as the one in the ISCXIDS 2012 and CICIDS 2017 datasets) was confirmed.
- Forecasting algorithms are directly dependent on previous observations to make their predictions. Since network traffic contains natural, and possibly frequent, variations, its current state might be different from previous states. Hence, the forecasting algorithm needs to be retrained.
- This has been observed in our Chapter 7 experimental results, specifically for the ISCXIDS 2012 and CICIDS 2017 datasets and their legitimate traffic. ForChaos first-layer had to be retrained at approximately every 13000-3300 seconds.
- On the other hand, regarding intrusion detection, the combination of a high in variation network traffic (experimental results presented in this chapter) with a forecasting model that has not been re-trained for quite some time can potentially lead to the forecasting algorithm not being able to identify malicious behaviour at some point onwards. This was evident in the experiments of Chapter 6, where the 60-second window size, which was unable to fully detect the flooding attack when the attack start time was at the 4000th second (regardless of the attack duration). However, the 60-second window size demonstrated no issues in detecting the same attacks when the attack start time was earlier (1000th, 2000th and 3000th second).
- We also have to point out that the 60-second window size was unable to detect the flooding attack which consisted of 5 attackers (and not 10). Hence, we also believe that the lower impact rate played an important role along with the reasons already stated.

- Regarding the 40-second window size, beyond the need for re-training as effectively highlighted from the related studies already mentioned due to the traffic's natural variations, we also observe that the attack duration plays an important role for the specific window size. As already stated, a forecasting algorithm is directly dependent on previous observations to construct its predictions. The current observations monitored are going to be used for constructing the future forecasts.
- In our case, if an attack is initiated, then the first timestamp in which the attack takes place is going to be immediately flagged as something malicious. This is due to the present network state not being even remotely similar to what the algorithm has previously observed. As a result, the error produced (forecast – actual value) is large enough to produce a positive Lyapunov exponent.
- However, as the attack is in progress, more and more observations (which contain malicious traffic) are stored in the algorithm's list of previous observations. Hence, the forecasting algorithm at some point will start producing forecasts that are similar to the malicious traffic (since it has already monitored enough of this type of observations). Therefore, the error will not be large enough to produce a positive Lyapunov exponent anymore.
- That is the main reason the 40-second window size faces problems in identifying the whole duration of 400 and 500-second attacks, regardless of their attack start time, but not on attacks which consist of 200 and 300 seconds duration.
- Once again, we also have to point out that the 40-second window size faced issues with slow-rate attacks that consist of 10 apps for each attacker (and not the 20 apps for each attacker scenarios). Hence, and considering that the 60-second window size faced the same issue with the less intense flooding attack scenarios, the attack intensity plays an important role in detecting the attack for its entire duration.
- This, as observed, was not the case in the Chapter 7 experimental results, where ForChaos first-layer was able to detect the attacks, regardless of their duration and attack start time, due to their magnitude.
- In contrast to ICT networks, IoT traffic is unlikely to be so highly varied as related studies have supported [84], [157], [219], [207], [130], [69] , [201]. Despite ForChaosLR being an IDS specifically aimed for IoT networks (such as the smart home-smart grid IoT network), it is a positive and very reassuring outcome that when re-training

is introduced ForChaosLR is able to handle high variations of traffic, like the ones observed.

- With regards to the TP rates, we observe that although ForChaosLR does not achieve the highest TP rate, it produces equivalent or higher TP rate than the majority of the ML algorithms it was compared with. This indicates that although ForChaosLR is a less sophisticated algorithm compared to the ML algorithms used, its TP rate can be considered successful.
- In addition, we also observe that when the TP rates from the NS-3 binary and the CICIDS 2017 binary datasets from ForChaosLR are compared, the difference is no more than 0.3%. This clearly indicates that ForChaosLR is unaffected from the introduction of different variations of flooding and slow-rate DDoS attacks.
- Regarding the FP rates, we observe that ForChaosLR produces a substantial FP rate, specifically 4.53%. However, its FP rate is lower than the FP rate achieved by naive Bayes and all SVM algorithms.
- Furthermore, we observe that in this series of experiments, the FP rate by ForChaosLR is significantly higher than the equivalent rate for the NS-3 binary dataset (0.52%). We also have to point out that the majority of ML algorithms achieved higher FP rates than their equivalent NS-3 binary dataset FP rates (except decision tree, Bayesian network, KNN (Euclidean, Chebyshev and Manhattan K=1 and Manhattan K=3)). This indicates that the high variation of legitimate traffic had a negative impact on the majority of the ML algorithms, beyond ForChaosLR.
- In conclusion, we have to acknowledge that although ForChaosLR maintains a similarly high TP rate between the two datasets, its FP rate is substantially increased (along with naive Bayes's FP rate). The rest of the ML algorithms which had their FP rates increased had less difference. This demonstrates that even when LR second-layer is used or when re-training is used, the FP rate (although generally low) is higher than the majority of the FP rates achieved by the majority of the ML algorithms.
- However, we have to point out that ForChaosLR was primarily designed and implemented for IoT networks (such as the smart home-smart grid) where the variation of traffic is less likely to happen in frequency and in magnitude [84], [157], [219], [207], [130], [69], [201]. When ForChaosLR is evaluated in such datasets (such as our own NS-3 binary dataset) the FP rate is lower than most of the FP rates of the ML algorithms. In addition, when ForChaosLR is evaluated in the smart homes datasets

presented in this chapter, we observe that the FP rates are similar to the one achieved in our own NS-3 binary dataset, specifically 0.4% for the 2018 and 0.2% for the 2019 dataset. This shows that ForChaosLR successful in working well with IoT legitimate traffic, regardless of any temporary spikes that might occur.

f

Chapter 8

ForChaosLR IDS Accuracy Evaluation using Multi-Class Datasets

8.1 Overview and Motivation

In Chapters 6 and 7 we focused on evaluating ForChaosLR in differentiating DDoS from legitimate traffic using the NS-3 dataset from the smart home-smart grid network simulation and then real world datasets. In this chapter, we aim to evaluate ForChaosLR multi-layer IDS for multi-attack classification, an objective mentioned in Chapter 1.

The datasets used were multi-attack NS-3 and CICIDS 2017 datasets, Section 5.2.3 described in Chapter 5. It is important for an IDS to be able to perform multi-attack classification with high accuracy as different attacks require different mitigation techniques. In the literature [20], [216], [177], [221] the majority of multi-layer IDSs proposed consist of the second layer accepting malicious traffic identified from the first layer and further identifying it as a specific type of attack. Hence, the second layer in the majority of multi-layer IDSs proposed does not aim to improve accuracy but rather categorise incoming malicious traffic.

In a multi-attack classification we obviously aim to classify each attack correctly. However, even if a particular instance is not classified to the correct attack category it is still essential for it to not be misclassified as normal. Hence, to distinguish between these two cases: an attack instance correctly classified to the correct attack category and an attack instance classified to an incorrect attack but still classified as malicious we introduce two metrics, the TP-general rate (for the former case) and the TP-multi rate (for the latter case). We also compared ForChaosLR multi-layer IDS against different machine learning (ML)

algorithms. The relevant parameters used for the ML algorithms' training and construction can be found in Table 6.19.

The rest of the chapter is organised as follows:

- The experimental setup details section consists of the following:
 - Detailed descriptions of the multi-attack datasets used for the experimental evaluation of ForChaosLR multi-layer IDS.
 - Optimised values of the ForChaosLR multi-layer IDS's parameters after the training phase is finished.
- The results analysis and discussion of ForChaosLR multi-layers IDS using the NS-3 multi-attack datasets as well as its comparison with the ML algorithms chosen are presented.
- The results analysis and discussion of ForChaosLR multi-layers IDS using the CICIDS 2017 single category multi-attack datasets as well as its comparison with the ML algorithms chosen are presented.
- The results analysis and discussion of ForChaosLR multi-layers IDS using the CICIDS 2017 multi category multi-attack datasets as well as its comparison with the ML algorithms chosen are presented.

8.2 Experimental Setup Details for multi-attack datasets

We used the same NS-3 dataset as before but changed the attack label to flooding and slow-rate attack. As a result, there were 474 total number of flooding instances, 467 slow-rate instances, and 3228 normal instances. We used the SMOTE algorithm again, and our flooding instances was increased to 1896, and slowloris instances to 1868. The normal instances number was kept the same. We chose to exclude the mixed attacks from this set of experiments as there is no particular interest in investigating whether ForChaosLR multi-layer IDS would correctly classify such a type of malicious traffic. More information regarding the NS-3 multi-attack training and testing datasets are in Chapter 5, Section 5.2.3,

Using the CICIDS 2017 dataset, we carried out two types of evaluation experiments. In the first series of experiments we followed a similar approach to the NS-3 dataset construction, by dividing the attack instances to flooding and slow-rate. We call this dataset the single category multi-class dataset, described in Chapter 5 Section 5.2.3.

The second approach we followed is to classify the application layer attacks into specific types as defined by the specifications for the dataset. As a result, we expanded the class labels for this version of the dataset to flooding (constant), slowloris, slowhttptest, hulk and goldeneye. We call this dataset the multi category multi-class dataset and it is described in Chapter 5, Section 5.2.3.

For both these datasets we used the SMOTE algorithm to create additional synthetic instances for the malicious classes as the total number of them was not high enough to use for both training and testing. For the normal instances we used the "Monday" file but also increased their number using the SMOTE algorithm. The final version of single category multi-attack CICIDS 2017 training set consisted of 840 flooding (constant) instances, 1055 slowloris instances and 440 normal instances. The final version of the multi category multi-attack CICIDS 2017 training set consisted of 840 flooding (constant) instances, 924 slowloris instances, 462 slowhttptest instances, 720 hulk instances, 806 goldeneye instances and 880 normal instances.

In Table 8.1 we present ForChaosLR parameter values after training for the NS-3 multi-class dataset. In Table 8.2 we present ForChaosLR parameter values after training for the CICIDS 2017 single-category multi-class dataset. In Table 8.3 we present ForChaosLR parameter values after training for the CICIDS 2017 multi-category multi-class dataset.

Since we aim to classify incoming instances to more than two classes, a multi-class logistic regression training approach is adopted. During training, the number of logistic regression functions constructed is the same to the total number of classes. Hence, each class consists of its own logistic regression function and therefore, its own set of optimal coefficients. In the third column across the three tables, we enlist each coefficient for each class across all features. We used Weka [144] for all algorithms. We used the same training set as the one used for LR second-layer; we used the same test set as the one ForChaosLR.

8.3 Results Analysis for NS-3 multi-class dataset

In Figures 8.1, 8.2, 8.3, 8.4 and 8.5 we present the results for multi-class classification using the NS-3 multi-class dataset. Based on these results, we make the following observations:

- The results from this series of experiments followed the same trend as the results from the binary classification experimental results gained from the NS-3 binary class dataset.
- However, the FP rates for the majority of ML algorithms (except MLP neural network and SVM algorithms) decreased.

Table 8.1 ForChaosLR multi-layer variables multi-attack NS-3 Optimal Parameters

<i>Parameter</i>	<i>ForChaosLR layer</i>	<i>Value (flooding, slow-rate)</i>
RequestsNo	LR second-layer	(0.0206 ,0.0482)
AverageTimeBetwGetGet	LR second-layer	(-87.6634, -389.4809)
AverageTimeBetwOkGet	LR second-layer	(245.1389, 420.3207)
AverageTimeBetwOkOk	LR second-layer	(-83.0743, -38.5519)
PacketsNumber	LR second-layer	(0.0668, 0.0828)
PacketSize	LR second-layer	(0.1728, -0.0126)
AverageDataRate	LR second-layer	(-1004.2427, -2017.3752)
ParallelRequests	LR second-layer	(0.0017, -0.0017)
Intercept	LR second-layer	(-61.1468, -5.8274)
Ridge Value	LR second-layer	1.0E-8
Window Size	ForChaos first-layer	40 seconds
<i>a</i> value	ForChaos first-layer	0.1
Training Time	ForChaos first-layer	1000 seconds

Table 8.2 ForChaosLR multi-layer variables multi-attack CICIDS2017 single-category multi-class Optimal Parameters

<i>Parameter</i>	<i>ForChaosLR layer</i>	<i>Value (flooding, slow-rate)</i>
RequestsNo	LR second-layer	0.0951 ,0.016
AverageTimeBetwGetGet	LR second-layer	-2634.2221 , -236.5975
AverageTimeBetwOkGet	LR second-layer	651.2879, -6495.887
AverageTimeBetwOkOk	LR second-layer	-157.2252, -1237.0908
PacketsNumber	LR second-layer	-0.0104, -0.0029
PacketSize	LR second-layer	0.1672, -0.0008
AverageDataRate	LR second-layer	-29.6399, 4.936
ParallelRequests	LR second-layer	-0.0011, -0.0001
Intercept	LR second-layer	-313.8241, 4.8138
Ridge Value	LR second-layer	1.0E-8
Window Size	ForChaos first-layer	40 seconds
<i>a</i> value	ForChaos first-layer	0.1
Training Time	ForChaos first-layer	1000 seconds

Table 8.3 ForChaosLR multi-layer variables multi-attack CICIDS2017 multi-category multi-class Optimal Parameters

<i>Parameter</i>	<i>ForChaosLR layer</i>	<i>Value (flooding, slowloris, slowhttptest, hulk, goldeneye)</i>
RequestsNo	LR second-layer	0.0566, 0.0072, 0.0261, 0.0796, 0.0245
AverageTimeBetwGetGet	LR second-layer	-711.5098 , -3840.3198, -7394.4255, 252.0025, -27187.9539
AverageTimeBetwOkGet	LR second-layer	-2995.3802, -8832.5971, 2181.9354, -976.4418, 1918.1441
AverageTimeBetwOkOk	LR second-layer	589.1015, -2002.2373, -2610.0945, 310.0377, 954.1699
PacketsNumber	LR second-layer	-0.014, 0.0025, 0.0004, -0.014, -0.0035
PacketSize	LR second-layer	0.0443, -0.0053, 0.0115, -0.0107, 0.004
AverageDataRate	LR second-layer	18.7897, -5.19, -7.8423, 23.9854, 6.6174
ParallelRequests	LR second-layer	-0.0003, -0.0021, -0.002, 0.0004, -0
Intercept	LR second-layer	-107.5009, 43.0838, 28.7064, -163.2212, -8.0639
Ridge Value	LR second-layer	1.0E-8
Window Size	ForChaos first-layer	40 seconds
α value	ForChaos first-layer	0.1
Training Time	ForChaos first-layer	1000 seconds

- Hence, results suggest that a multi-label algorithmic model is more accurate in correctly classifying legitimate traffic than a binary algorithmic model

Bayesian Algorithms :

- Both Bayesian networks and naive Bayesian algorithms produce similar TP-general rates with the ones from the binary classification results of the NS-3 dataset. Hence, Bayesian models do not see any benefit in their TP-general rates when they construct an algorithmic model for detection.
- Their TP-multi rates are relatively high, however we notice that Bayesian networks algorithm's TP-multi rate is decreased by approximately 2% while naive Bayesian algorithm's TP-multi rate is only decreased by 0.5%.
- Comparing the FP rates for both Bayesian algorithms were negatively impacted, as their respective rates were significantly increased, compared to the NS-3 binary dataset results. Hence, we conclude that a multi-class approach has a negative impact for both Bayesian algorithms with regards to the FP rates.

Neural Network Algorithms :

- In contrast to the NS-3 binary dataset results where MLP(0.3) produced a higher TP-general rate from MLP(0.5) now it is reversed. MLP(0.5) generates a higher TP-general rate with an approximate 1% difference from MLP(0.3), a result which is not highly significant though.
- Furthermore, MLP(0.5) algorithm's TP-general rate is increased by approximately 1.5% from the binary-classification experimental results. In contrast, MLP(0.3) achieves a lower TP-general by approximately 1% and as a result, it is not benefited from the multi-class training.
- Hence, a higher learning rate combined with multi-class training gives higher TP-general and TP-multi rates.
- In contrast to the rest of the ML algorithms and our ForChaosLR, both neural network algorithms generate higher FP rates, specifically approximately 2% for MLP(0.3) and approximately 1.5% for MLP(0.5).
- Therefore, although multi-class training benefited MLP(0.5) in increasing its TP-general, it negatively impacted both MLP algorithms in their accuracy.

SVM Algorithms :

- Comparing the SVM algorithms TP-general results with the respective ones from the NS-3 binary dataset, we conclude that the multi-class training was not benefited the more complex kernels of the SVM algorithm. However, the difference between the TP-general rates is no more than 1.3%. Comparing the TP-general rates with the TP-multi rates the decrease is only approximately 0.1%.
- Regarding FP rates, SVM(RBF) algorithm generates the lowest FP rate which was also observed in the NS-3 binary dataset results.
- In contrast to the NS-3 binary dataset results, SVM(Linear) generates the least FP rate compared to SVM(Quadratic) and SVM(Cubic). However, their difference is not significant.
- Concluding, multi-class training was not beneficial for the SVM algorithms.
- However the difference between these results is not higher than approximately 2% regarding TP rates and not higher than 1.4% regarding FP rates when compared to each other.

KNN Algorithms :

- In this series of experiments, the KNN algorithms follow the same trends as the ones from the NS-3 binary dataset results.
- In detail, KNN (K=1, Filtered) once again achieves the highest TP-general rate, 96.161%. The same algorithm achieved the highest TP-general rate in the NS-3 binary dataset.
- Furthermore, once again an increase in K number decreases the TP-general rate across all KNN algorithms except the KNN algorithm that uses Filtered distance. However, this decrease is no higher than approximately 0.2% which is not highly significant.
- Interestingly, all of KNN algorithms' TP-multi rates are not decreased when compared to their TP-general rates.
- Regarding FP rates, the results are approximately unaffected compared to the NS-3 binary dataset results with no significant differences.

- Furthermore, once again when K is increased so does the FP rate across all KNN algorithms, a trend which is also present in the NS-3 binary dataset results. Once again, the difference is no more than 1.1% therefore, not highly significant.

Tree Algorithms :

- In contrast to the rest of the ML algorithms, both decision tree and random forest algorithm achieve approximately the same results in TP-general rates when a multi-class training approach is followed.
- Decision tree algorithm once again generates the highest TP-general rate, a trend also observed in the NS-3 binary dataset results. However, their difference is less than 0.7% which is not a highly significant rate.
- In contrast, their TP-multi rates are approximately 2-2.5% lower than their TP-general rates. Decision tree algorithm generates a higher TP-multi rate than random forest algorithm but only by approximately 0.07% which is not a significant difference.
- Regarding FP rates we observe a decrease of approximately 1% when compared to the results from the binary classification. Once again random forest algorithm achieves a lower FP rate than decision tree algorithm, possibly due to its higher complexity during the training phase.
- Hence, we conclude that random forest algorithm is more robust against FP rates compared to the decision tree algorithm.

ForChaosLR results and comparison with ML algorithms :

- ForChaosLR's TP-general rate is 97.988%, approximately 0.75% higher than the TP rate achieved using the NS-3 binary dataset. This indicates that the LR second-layer model is unaffected by the introduction of additional labels and maintains a high TP rate.
- Compared to the TP-general rates of the ML algorithms, we observe that ForChaosLR, achieved a higher TP rate than the majority of the ML algorithms, except Bayesian networks, naive Bayesian and decision tree algorithms.
- Furthermore, observing the TP-multi rates, ForChaosLR achieved a higher TP-multi rate than the majority of the ML algorithms, except naive Bayes, KNN(Filtered), SVM (Cubic and Linear) and MLP(0.5).

- Based on the two previous points, we conclude that although ForChaosLR did not achieve the highest TP-general and/or TP-multi rate, it is one of the most successful algorithms in not only detecting malicious DDoS traffic but also correctly classifying it.
- Regarding FP rates, ForChaosLR multi-layer IDS's is significantly decreased from 0.520% (NS-3 binary dataset) to 0.056%. This indicates that a multi-class training approach of the LR second-layer had a positive impact on the overall accuracy of ForChaosLR. We believe the reason for this improvement is on the nature of the attacks. Although these two attacks have the same objective (make the target unavailable to legitimate requests), their overall network traffic behaviour is not similar. In general, flooding generates higher rates of traffic than slow-rate which is considered a stealthier approach [19]. As a result, attempting to have these two different types of behaviour labelled the same might cause issues. As a result, a multi-class labelling had a positive impact on the FP rates for not only ForChaosLR but for the majority of the ML algorithms as well. Their equivalent FP rates are shown to be decreased by substantially. Hence, we conclude that a multi-class algorithmic model is able to decrease the FP rate to a greater extent compared to the binary constructed model.
- Concluding, although ForChaosLR does not achieve the highest TP rate (general or multi), it is one of the most successful algorithms in detecting malicious traffic and correctly classifying it based on the results obtained. In addition, in this series of experiments, ForChaosLR achieves the lowest FP rate compared to the ML algorithms, highlighting that a multi-class training approach has been beneficial to all the ML algorithms (including ForChaosLR) as they all decreased their FP rates (when compared to the equivalent NS-3 binary dataset FP rates).

8.4 Results Analysis for CICSIDS2017 Single-Category Multi-class Dataset

The results of the CICIDS 2017 multi-class single category dataset are demonstrated in Figures 8.6, 8.7, 8.8, 8.9 and 8.10. We present our main observations below.

Bayesian Algorithms :

- A multi-class problem is harder to be solved than a binary-class one, as the training is more complex and multi-dimensional. Based on the results, we understand that

Bayesian networks algorithm faces issues in classifying incoming traffic a DDoS attack when trained using the multi-class approach.

- Bayesian networks algorithm's TP-general rate is decreased by approximately 1.5% in comparison to the CICIDS 2017 binary classification dataset results.
- Bayesian networks algorithm's the TP-multi rate is even further decreased but by only approximately 1.5%.
- In contrast, naive Bayes algorithm's TP-general rates are improved as it achieves a 100%, more than 3% increase from the CICIDS 2017 binary classification dataset results. Its TP-multi rate is decreased but by only approximately 1.5%.
- However, naive Bayes algorithm fails to completely identify legitimate traffic in contrast to Bayesian networks algorithm which generates a 4.467% FP rate, an acceptable result.
- Hence, although naive Bayes algorithm's TP rates (both general and multi) are quite high, are of no use as it is not able to identify legitimate traffic and was not benefited from the multi-attack training .

Neural Network Algorithms :

- There is an improvement in both algorithms (0.3 and 0.5 learning rates) from the CICIDS 2017 binary classification dataset results in both TP rates by approximately 2%.
- Both neural network models achieved the same TP-general and TP-multi rates.
- The FP rates were nearly doubled for both neural network models when compared to the CICIDS 2017 binary dataset results.
- This indicates that a multi-class approach had a negative impact with regards to the FP rates.

SVM Algorithms :

- SVM(Linear) and SVM(RBF) performed worse in this series of experiments than the CICIDS 2017 binary classification dataset results in TP-general rates, having a decrease of approximately 7% and 9% prospectively.

- On the other hand, SVM(Quadratic) and SVM(Cubic) algorithms had an approximate increase of 2%.
- It is evident that the multi-attack training approach had a positive impact on the quadratic and cubic kernel models but not on the linear and RBF ones with respect to TP-general rates. This can be justified as a linear kernel is unlikely to produce better results than a higher-degree kernel for multi-class problems.
- SVM(RBF) also appears to be ineffective when the multi-attack training approach is used. Examining the results further, we also observe that SVM(RBF) generates only a 71.538% TP-multi rate.
- The rest of the SVM algorithms have a slight decrease on their TP-multi rates when compared to the TP-general rates which is not a significant rate though.
- Regarding FP rates, all four SVM algorithms have high FP rates as opposed to their respective results in the CICIDS 2017 binary classification dataset.
- SVM(Linear) had the lowest FP rate followed by SVM(RBF), SVM(Quadratic) and then SVM(Cubic) all three of them resulting in nearly double or more than double the former's FP rate.
- Based on the results, we conclude that the multi-attack training approach was quite ineffective in achieving low FP rates for all versions of the SVM algorithm.
- Although, the Quadratic and Cubic kernel SVM algorithms slightly improve their TP-general rates, the trade-off between the TP and the FP rate is high.
- We believe the reason SVM(Linear) generates the least FP rate is because its linearity is effective in detecting legitimate traffic as opposed to the RBF, quadratic, cubic and kernel SVM algorithms. However, the latter two are effective in detecting flooding and slow-rate traffic since they use a higher degree kernel which assists in effective DDoS traffic classification.

KNN Algorithms :

- KNN with K=1 and Filtered distance generates the best TP-general and TP-multi rates with an accuracy of 99.231% as opposed to the results from the CICIDS 2017 binary dataset where KNN Euclidean and Manhattan (regardless of K number) performed best.

- When K is increased and filtered Euclidean distance is used, the TP-general and TP-multi rates are slightly decreased by 0.769%, a difference which is not significant.
- The TP-general and TP-multi rates remain unaffected.
- Euclidean and Manhattan distance achieved a full 100% TP-general rate and now we see a decrease of 1.538%.
- The differences when comparing the results from the two datasets (CICIDS 2017 binary and multi-class datasets) are not highly significant. However, we believe that KNN Filtered and Chebyshev distance benefited from the multi-class labelling approach, hence why they produce higher TP-general results.
- On the other hand, Euclidean and Manhattan distance KNN versions performed worse in the CICIDS 2017 multi-class dataset than in the CICIDS 2017 binary dataset however, their differences are not significant.
- We observe an increase in FP rates for this series of experiments as opposed to the results achieved using the CICIDS 2017 binary classification dataset. This observation is highlighted when K number is increased.
- In detail, when Euclidean, Manhattan and Chebyshev distance are used, the FP rate almost doubles when K is increased from 1 to 3, a difference which was not present in the results from the CICIDS 2017 binary classification dataset.
- KNN filtered is unaffected by this trend and generates only a slightly higher FP rate when K is set to 3 from 1, with only 0.343% difference. This indicate to us that multi-class labelling does not assist in classifying incoming traffic as legitimate correctly when the K number is increased.

Tree Algorithms :

- Decision tree algorithm's TP-general rate was decreased by more than 2% as opposed to random forest algorithm's which remained unaffected and also when compared to the CICIDS 2017 equivalent results.
- Decision tree's algorithm TP-multi rate is lower than random forest's by more than 5%. Based on the two previous points made, we draw two important conclusions:
 - 1) Multi-class training approach had a negative impact on the decision tree algorithm as the TP-general rate was decreased.

- We assume that multi-class training caused issues in the correct construction of the decision tree model.
 - In contrast, random forest algorithm remain unaffected in terms of TP-general rate as it maintains the same result.
 - 2) Random Forest algorithm clearly surpasses the decision tree algorithm in TP-multi rate with an approximate difference of 6%.
 - We believe the reason for both of these conclusions is due to the nature of the random forest algorithm which trains multiple tree algorithmic models in order to improve accuracy and minimise bias, which is clearly proved effective in our case.
- The FP results are slightly increased for both algorithms compared to the results achieved in CICIDS 2017 binary classification dataset. Their difference however, cannot be considered significant.
 - In contrast to the CICIDS 2017 binary dataset FP rates results, in this series of experiments random forest algorithm achieves a lower FP rate than decision tree algorithm FP rate. Hence, a multi-class training approach has clearly benefited the random forest algorithm.
 - Once again we believe that the training of multiple tree models with different parts of the training set used by the random forest algorithm was proved effective as opposed to the Decision Tree algorithm.

ForChaosLR results and comparison with ML algorithms :

- Firstly, ForChaosLR's TP-general and TP-multi rates are identical (97.744%), which indicates that ForChaosLR can classify incoming malicious traffic to the correct attack type with high success.
- When compared with its equivalent results of the CICIDS 2017 binary dataset, we observe that the TP(general) rate remained unaffected from the introduction of additional labels. The same observations was valid with ForChaosLR's NS-3 binary and multi-class datasets' TP rates. This demonstrates ForChaosLR's robustness against the introduction of new labels during training.
- When compared with the ML algorithms, we observe that ForChaosLR does not achieve a higher TP-general rate than most of the ML algorithms, except for decision tree, SVM(RBF) and Bayesian network algorithms.

- In contrast, when comparing the equivalent TP-multi rates, ForChaosLR achieves a higher rate than many of the ML algorithms, specifically: Bayesian network, both MLP neural networks, all SVM, decision tree and random forest algorithms. This demonstrates that although ForChaosLR is worse in detecting malicious traffic compared to the majority of the ML algorithms, it is better in correctly classifying malicious traffic to a specific attack type than most of them.
- With regards to FP rates, we observe that ForChaosLR achieves an increased FP rate, compared to its equivalent FP rate with the CICIDS 2017 binary dataset, with a difference of approximately 1.5%. Although the difference is not substantially significant, we have to acknowledge that the introduction of additional labels in this series of experiments was not beneficial to the FP rates. This observation was also applicable to the rest of the ML algorithms which achieved a higher FP rate compared to their equivalent FP rates from the binary CICIDS 2017 dataset. Despite this, ForChaosLR achieves a lower FP rate than all naive Bayesian, SVM algorithms and both MLP neural networks.
- In summary, based on the results obtained regarding both TP(general) and FP rates, ForChaosLR is a more successful algorithm than SVM algorithms, as their FP rates are 3.5-14% higher than ForChaosLR's. In addition, SVM(RBF) has a nearly 8% difference in TP(general) rate. In addition, ForChaosLR achieves substantially better results than naive Bayesian as the latter has an unacceptable 100% FP rate. This trend has been evident in the CICIDS 2017 binary dataset results.
- With regards to the rest of the ML algorithms, in some cases ForChaosLR achieves a slightly higher TP rate than some algorithms (decision tree and Bayesian network) and in some cases it achieves a slightly lower FP rate than some other algorithms (both MLP neural networks).
- However, we have to acknowledge that ForChaosLR has not been as successful in this series of experiments as it was in the CICIDS 2017 binary or the NS-3 multi-class dataset results obtained. We believe the reason for the NS-3 multi-class dataset is because of the high variation of legitimate ICT traffic compared to the IoT network traffic. The reason for the CICIDS 2017 binary is because ForChaosLR is not benefited from a multi-class approach against a highly varied traffic dataset.

8.5 Results Analysis for CICSIDS2017 Multi-Category Multi-Class Dataset

In Figures 8.11 and 8.12, 8.13, 8.14, 8.15 for CICSIDS2017 multi category multi-class dataset. Based on the results, we draw the following set of general conclusions:

- The majority of the ML algorithms, with a few exceptions, improved the three main rates: TP-general, TP-multi and FP rates. This shows that a further labelling of the application layer DDoS attacks into different categories benefited the training procedure for the algorithm, assisting them into constructing more accurate models and therefore improve their classification rates.
- Although application layer DDoS share a common objective, they achieve this in different ways. Hence, a more detailed categorisation of the different application layer DDoS attacks gives a more clear view on how each attack behaves explicitly and how an algorithm should construct its model for more accuracy.

Proceeding, we compare the different ML algorithms and describe our most important observations.

Bayesian Algorithms :

- Bayesian networks algorithm's TP-general rate is increased by approximately 2.5% in comparison to the CICSIDS 2017 binary class classification dataset results.
- Although a multi-class problem is harder to be solved than a binary-class model, it is evident that the multi-category multi-class labelling has a clear benefit on the Bayesian networks algorithm.
- This observation is also reflected on its TP-multi which is improved by more than 1.5% even if in this series of experiments the number of classification labels is increased, therefore making it harder for an algorithm to be accurate. We believe that the multi-labelling of the different slow-rate attacks assisted the Bayesian networks algorithm into constructing a more accurate model.
- On the other hand, naive Bayes algorithm suffered a small decrease in both TP-general and TP-multi rates, with less than 0.8% for the former and less than 1% for the latter. These differences are not highly significant though, and we consider the results successful and highly accurate.

- Regarding FP rates, it is evident that the multi-labelling of the slow-rate attacks benefited greatly both algorithms.
- Bayesian networks algorithm improved its FP rate by a nearly 1.5% and naive Bayes by more than 80% which is a highly significant difference.
- Although naive Bayesian's FP rate is still high, it significantly improves in classifying incoming legitimate traffic correctly. However, across all three main rates we conclude that Bayesian networks algorithm is a better solution than naive Bayesian, as it generates a much lower FP rate and their differences in TP-general and TP-multi as not significant.

Neural Network Algorithms :

- There is an improvement in MLP(0.5) model from the CICIDS 2017 binary classification dataset results in both TP rates by nearly 1%. The MLP (0.3) has its TP-general rate remain the same and the TP-multi rate increased by nearly 1%.
- There is a decrease in FP rates for both MLP models, approximately 0.4% for MLP(0.3) and approximately 1% for MLP(0.5).
- In general, we observe an improvement across all three main rates, TP-general, TP-multi and FP .
- Therefore, we conclude that the multi-labelling of slow-rate attacks during training had a positive impact on the following:
 - General detection of DDoS traffic.
 - Correct classification of DDoS traffic in different attack types.
 - Reduction of FP rates.

SVM Algorithms :

- In general, the results either improved or remained the same in terms of TP-general and TP-multi rates. This indicates that the multi-class labelling of slow-rate attacks training assisted all the different versions of the SVM algorithm in constructing more accurate models.
- SVM(Linear) algorithm's TP-general rate improved by nearly 8%, a notable difference and SVM(RBF) algorithm's TP-general rate improved by more than 9%. Their TP-multi rates are also significantly improved by more than 4.5% and 19% respectively.

- In contrast, SVM(Quadratic) and SVM(Cubic) algorithms' TP-general rates remained unchanged and their TP-multi rates was decreased by approximately 3-4%.
- SVM(Linear) algorithm's FP rate demonstrated approximately a 0.4% increase which is not highly significant.
- SVM(Quadratic) and SVM(Cubic) demonstrated a high reduction with approximately 11%.
- In contrast, SVM(RBF) algorithm's FP rate was increased by approximately 20%.
- We conclude that the multi-category multi-attack training approach proved to be beneficial in the following ways:
 - The TP-general rates for SVM(Linear) and SVM(RBF) increased.
 - The FP rates for SVM(Quadratic) and SVM(Cubic) decreased.
- However, the trade-offs of TP-general, TP-multi and FP rates between the different versions of the SVM algorithm are quite significant.
- Despite this, we observe a consistency between the two datasets' results of the SVM(Quadratic) and SVM(Cubic) algorithms for the TP-general and TP-multi rates as the former has not demonstrated any changes in rate and the latter an approximate 3-3.9% decrease.
- We also observe a consistency of SVM(Linear) and its FP rate which is approximately the same with a minor 0.4% difference. SVM(RBF) remains highly unstable as the results achieved on the two series of experiments have notable differences between TP-general, TP-multi and FP rates.

KNN Algorithms :

- KNN with K=1 and filtered distance achieves the best TP-general and TP-multi rates with an accuracy of 100%. However, when K is increased and filtered distance is used, the TP-general and TP-multi rates are slightly decreased by approximately 1.5%.
- When K is increased, the TP-general and TP-multi rates are decreased in all the other KNN algorithms however, however this change is not significant. This comes in contrast to the previous results from the single category multi-class CICSIDS 2017 dataset which showed no change in TP-general rates regardless of the K number.

- Hence, we conclude that the introduction of additional labels has not significantly improved the KNN algorithms' TP-general and TP-multi rates.
- Regarding FP rates, the FP rate also increased with the value of K. Specifically, the FP rate is almost doubled in Euclidean and Chebyshev KNN algorithms, a trend which was present in the single category multi-class CICIDS 2017 dataset. This trend was also present in the binary 2017 dataset.

Tree Algorithms :

- Decision tree algorithm's TP-general rate was increased by nearly 2% and random forest by nearly 1.5%.
- However, decision tree algorithm's TP-multi rate was increased by nearly 4% and random forest's by approximately 1.5%.
- Based on these results, multi-category multi-class training approach had a positive impact on both algorithms. The possible further labelling of the various application layer DDoS attacks assisted them in constructing more accurate tree-based algorithmic models instead of constructing general rules that might not be applicable to the different types of application layer DDoS attacks.
- The FP results are slightly increased for the decision tree algorithm compared to the results achieved in CICIDS 2017 single category multi-class dataset. This was expected, as this is more complex problem and therefore more likely for the algorithms to generate a higher error. However, the FP rate difference is not highly significant, approximately 0.7%.
- The FP rate for the random forest algorithm remained unaffected.

ForChaosLR results and comparison with ML algorithms :

- Regarding ForChaosLR's TP-general rate remains the same as before, 97.744%. This indicates that its algorithmic model is unaffected by the introduction of additional labels during training. This trend was evident in the majority of the ML algorithms which had their TP-general rates either unaffected or slightly increased by approximately 1%. The only exception were the SVM(Linear) and SVM(RBF) which had a 9% increase. Hence, the introduction of additional labels was beneficial some ML algorithms while other remained unaffected, such as ForChaosLR.

- However, the TP-multi rate is decreased by approximately 2.3%. This trend of TP-multi rates being decreased was evident in the majority of the ML algorithms used. Hence, we observe that most of the ML algorithms along with ForChaosLR were not benefited from the introduction of additional labels during training.
- Based on the results obtained, ForChaosLR achieved a higher TP-general rate than only decision tree and KNN(Chebyshev and $K=3$). This clearly indicates that the introduction of additional labels has significantly improved the ML algorithms TP rates but not ForChaosLR's.
- With regards to the FP rates, ForChaosLR's rate was insignificantly decreased from 5.923% to 5.882% (single attack v.s. multi-attack). This highlights that the introduction of additional labels has mainly left the rates of ForChaosLR unaffected. The same conclusion was accountable for some of the ML algorithms, specifically: random forest, KNN (Filtered) and KNN (Manhattan $K=3$). Some ML algorithms had a slight increase (SVM(Linear), SVM(RBF), KNN(Euclidean and Chebyshev) and decision tree) while the rest improved their FP rates.
- Nevertheless, comparing the FP rates, we observe that ForChaosLR achieves a lower FP rate than naive Bayes, MLP(0.3), all SVM algorithms and KNN($K=3$ Chebyshev and Euclidean distances).
- Concluding, the introduction of additional labels has not improved ForChaosLR's accuracy in both TP and FP rates. It is evident that in more complex problems (multi-class classification problems) logistic regression does not perform as well as more complex algorithms such as MLP neural networks, random forest and Bayesian networks.

8.6 Summary

In this chapter, we presented all of the experimental results for ForChaosLR multi-layer IDS using the NS-3 and real-world multi-class datasets presented in Chapter 5. It is important for an IDS to not only distinguish between legitimate and malicious traffic but to also correctly classify the latter to a specific attack category. This can be of valuable assistance to any mitigation and response countermeasures deployed. Also, we needed to investigate whether a possible multi-class training approach could potentially improve the TP rate and maintain or eventually decrease the FP rate of ForChaosLR. In addition, we were interested on how the different ML algorithms would behave on such scenarios.

From the results presented in this chapter, we observe that ForChaosLR's TP rate has slightly increased when comparing the equivalent TP rates from the binary datasets with the equivalent multi-class rates. However, this increase (0.7% for NS-3 dataset and 0.05% for CICIDS 2017 dataset) is not considered significant. With regards to ForChaosLR's FP rates, we observe that the FP rate for the NS-3 dataset was decreased (approximately 0.46% difference) but for the CICIDS 2017 datasets was increased (approximately 1.4% difference).

Hence, we conclude that a multi-class training approach was not, in overall, beneficial for ForChaosLR, although it managed to maintain the high TP rates and either decrease or not substantially increase the FP rates. This was particularly highlighted through the comparison of the results for the CICIDS 2017 single class with the CICIDS 2017 multi class results. The increase of the labels did not improve either the TP or the FP rates.

Furthermore, we should acknowledge that while ForChaosLR's accuracy was not improved (or slightly decreased from a FP rate of view). In contrast, the ML algorithms' accuracy rates were improved for the CICIDS 2017 single and multi category datasets. This improvement in either TP or FP rates managed to surpass the respective rates of ForChaosLR. We believe this is because more complex problems (multi-class problems) can be solved more efficiently with a more sophisticated algorithmic approach.

In contrast, this was not the case with the NS-3 multi-class dataset, where ForChaosLR achieved the lowest FP rate and one of the highest TP rates (lower only from Bayesian networks, naive Bayesian and decision tree algorithms). We believe this is due to the high variation of legitimate network traffic combined with the different types of flooding and slow-rate DDoS attacks present in the two CICIDS 2017 multi-class datasets. Hence why ForChaosLR's FP rate in both series of experiments (CICIDS 2017 single and multi category multi-class datasets) was substantially increased compared to the FP rate from the NS-3 multi-class dataset.

Concluding, regarding both multi-class CICIDS 2017 datasets, we observe that although ForChaosLR produces good accuracy results it does not surpass most of the ML algorithms' respective results, with some exceptions. Specifically, these exceptions are Bayesian networks, SVM(Linear and RBF) and tree-based algorithms for CICIDS 2017 single category and KNN (K=3 Chebyshev and Manhattan) for the CICIDS 2017 multi-category datasets.

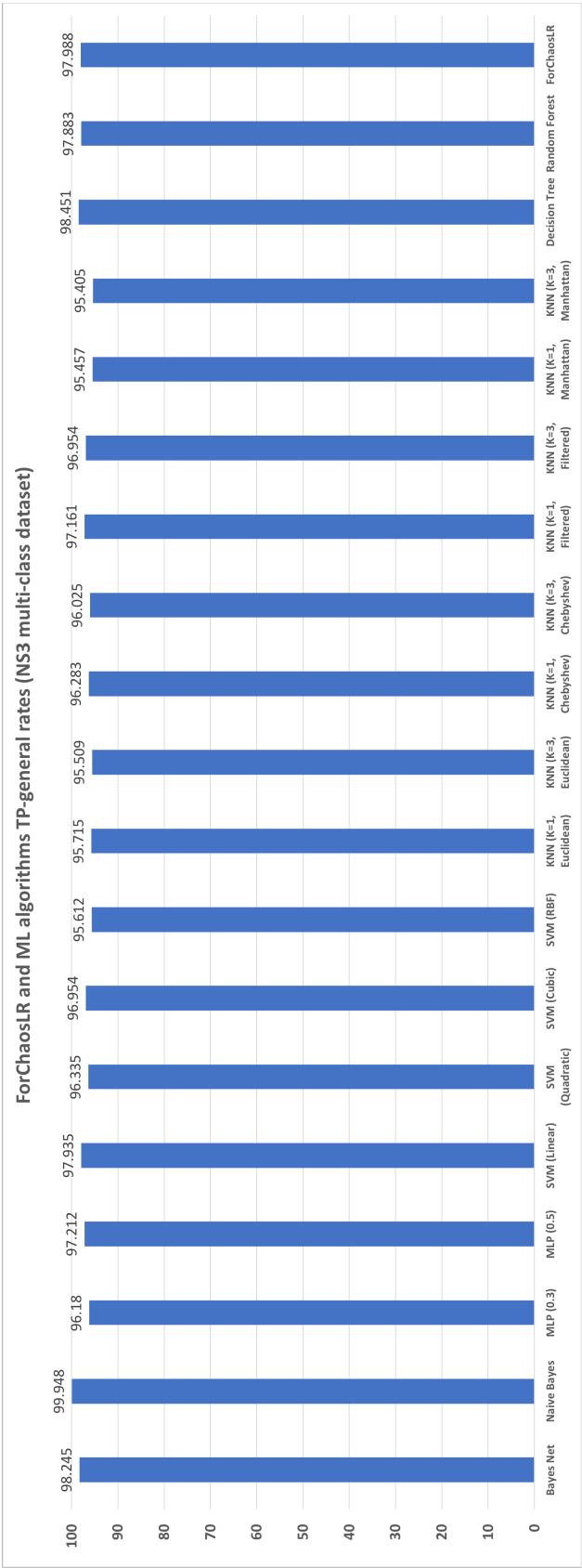


Fig. 8.1 ForChaosLR and ML algorithms TP rates for multi-class dataset

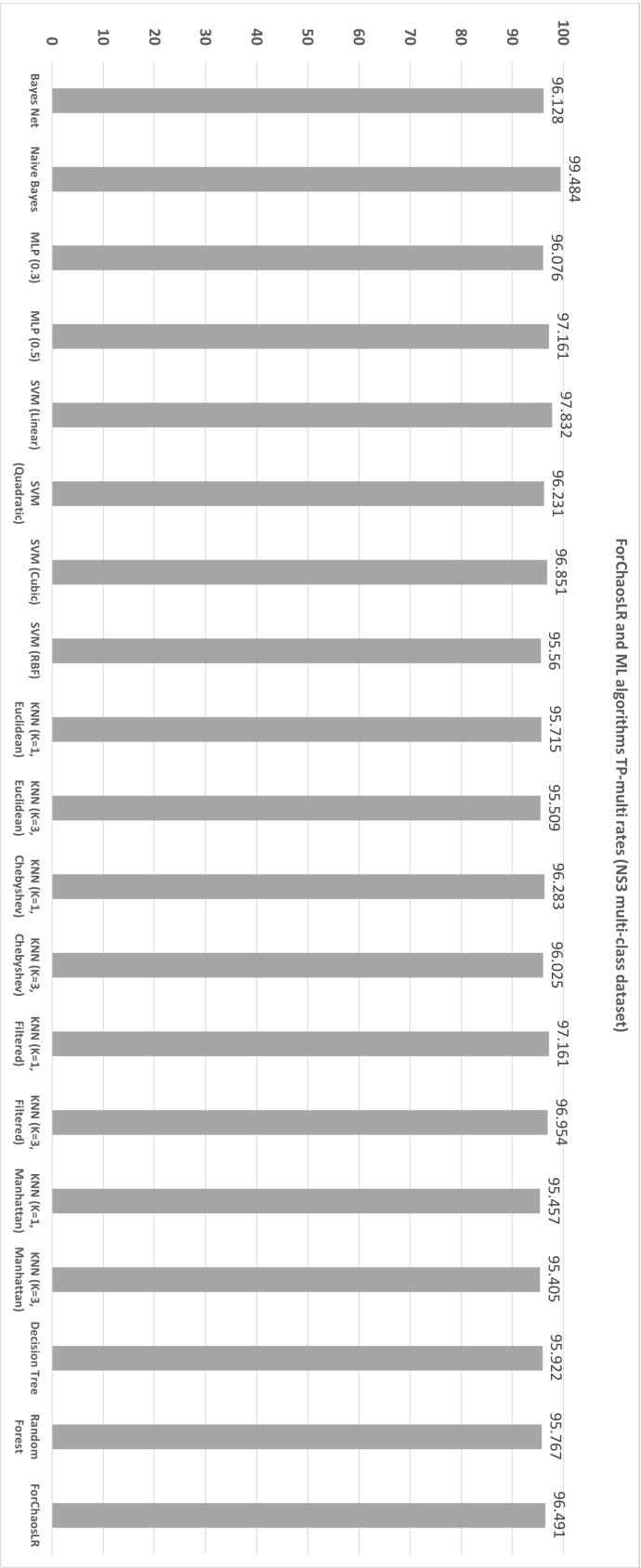


Fig. 8.2 ForChaosLR and ML algorithms TP-multi rates for multi-class dataset

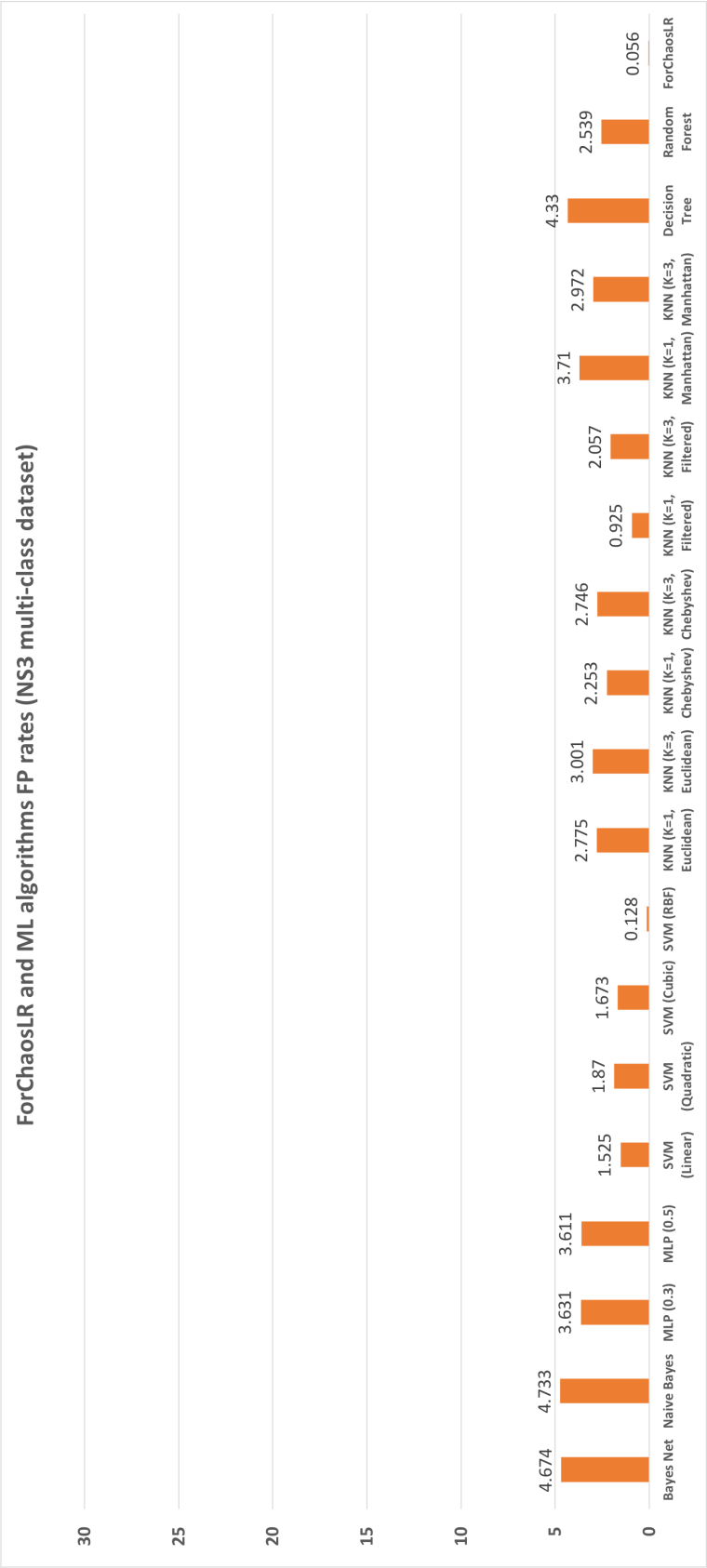


Fig. 8.3 ForChaosLR and ML algorithms FP rates for multi-class dataset

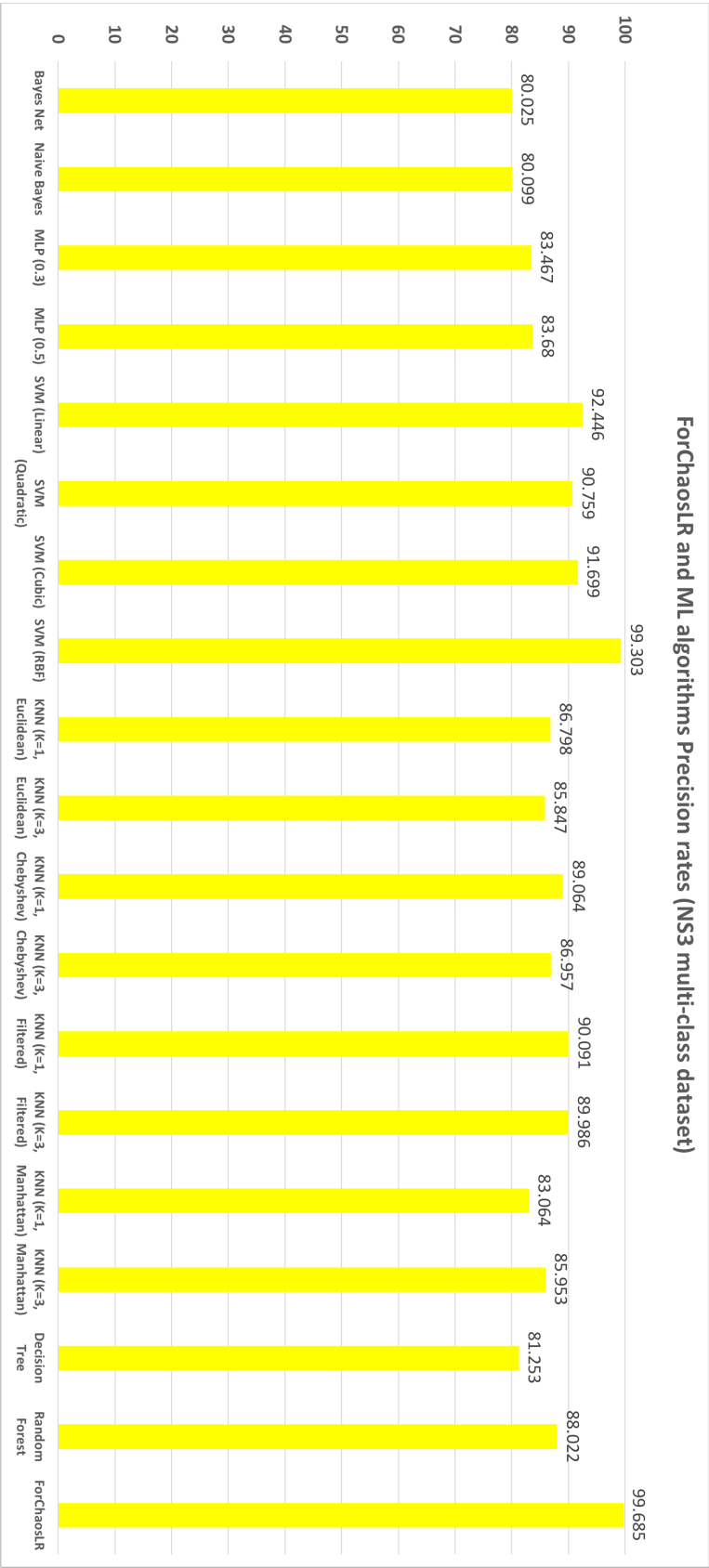


Fig. 8.4 ForChaosLR and ML algorithms Precision rates for NS-3 multi-class dataset

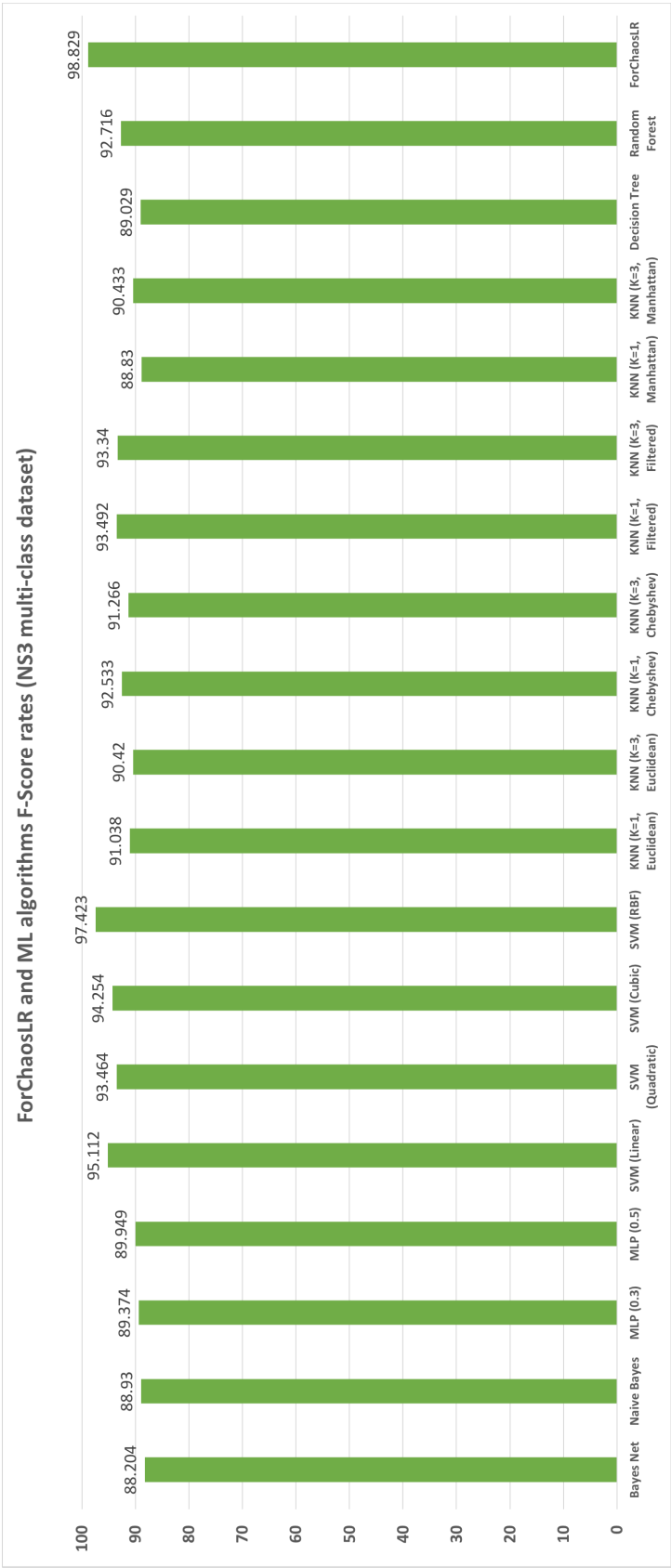


Fig. 8.5 ForChaosLR and ML algorithms F-Score rates for NS-3 multi-class dataset

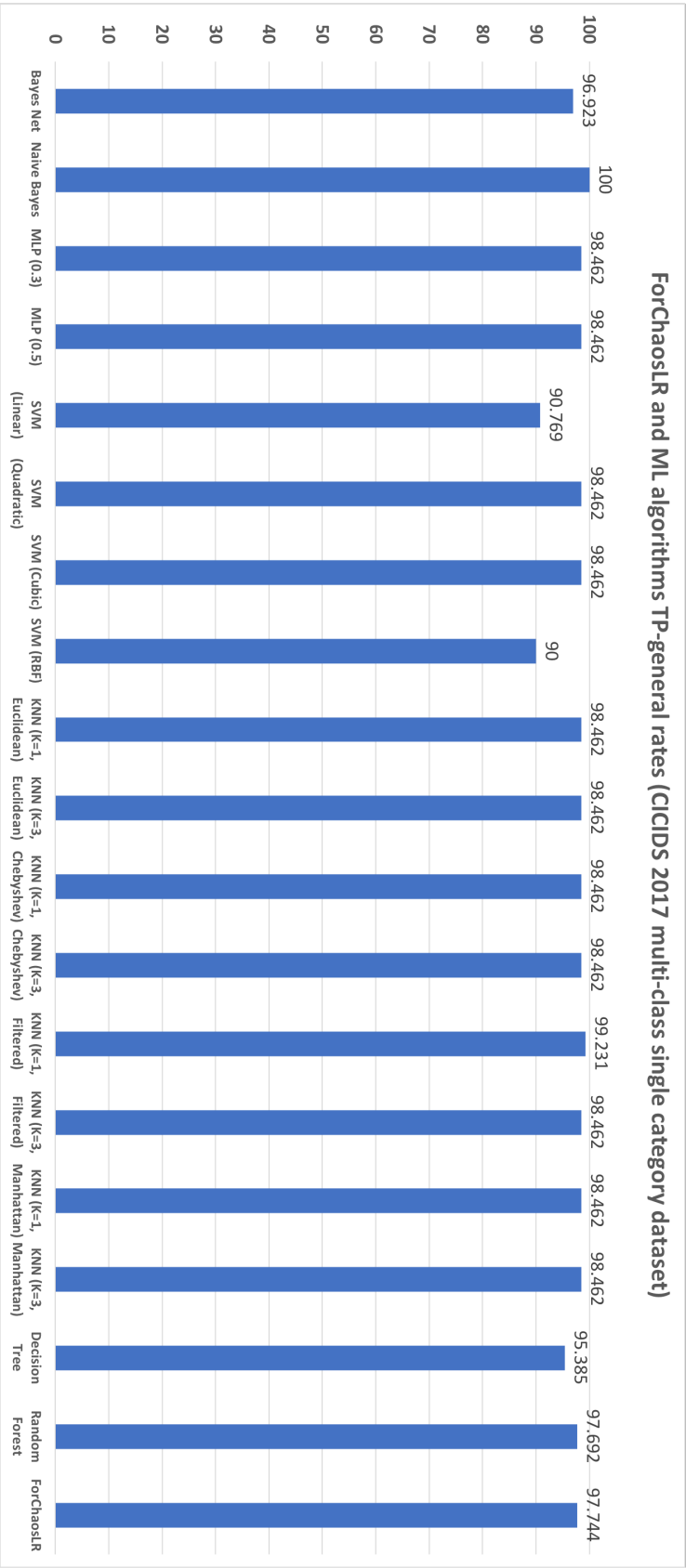


Fig. 8.6 ForChaosLR and ML algorithms TP rates for CICIDS2017 single category multi-class dataset

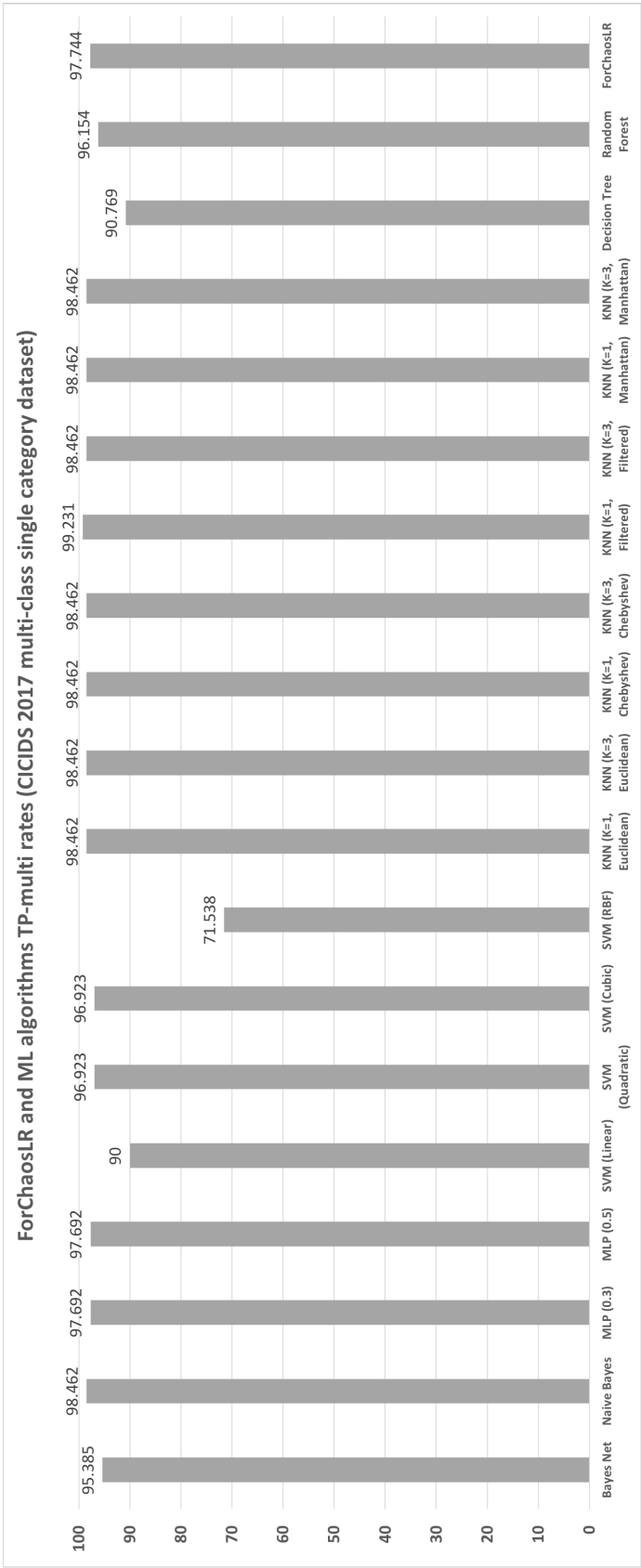


Fig. 8.7 ForChaosLR and ML algorithms TP-multi rates for CICIDS2017 single category multi-class dataset

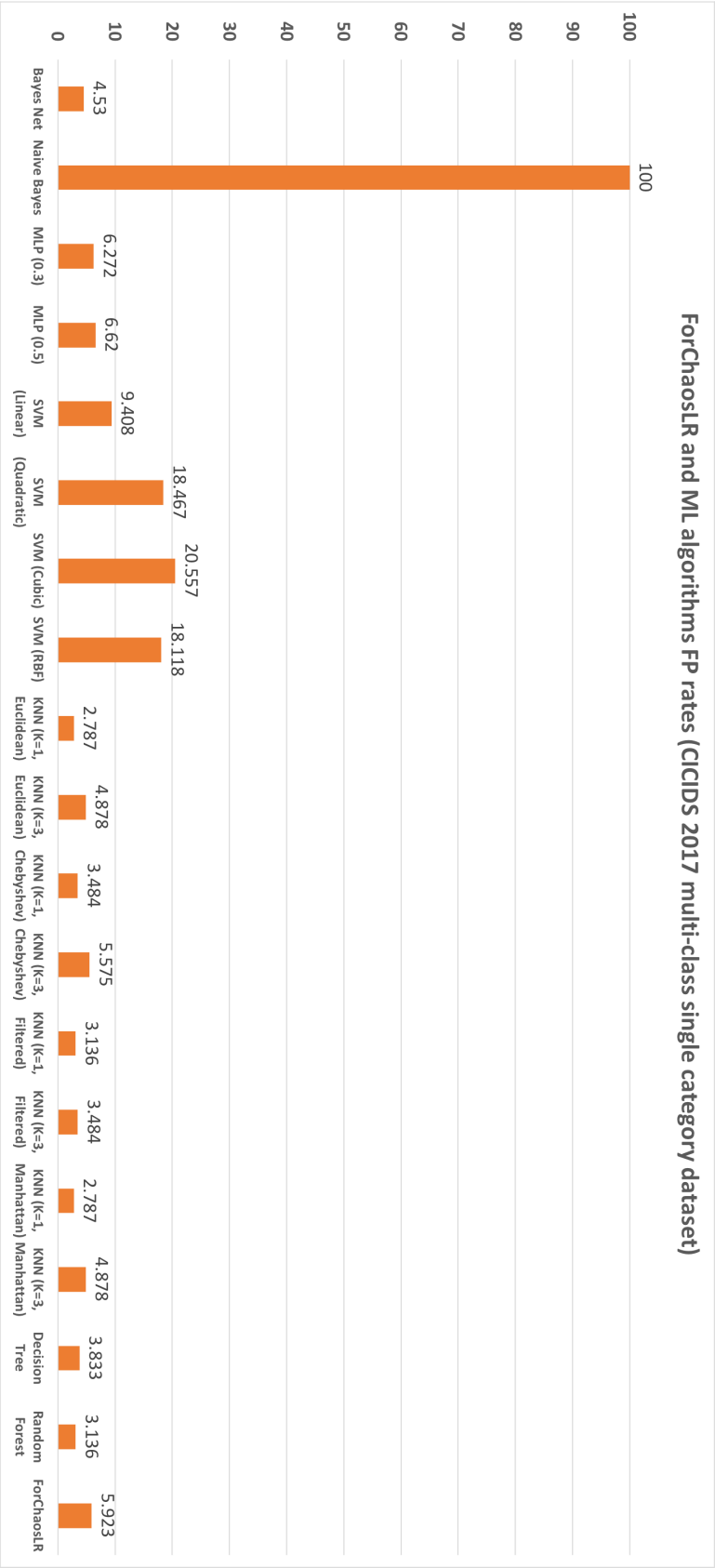


Fig. 8.8 ForChaosLR and ML algorithms FP rates for CICIDS2017 single category multi-class dataset

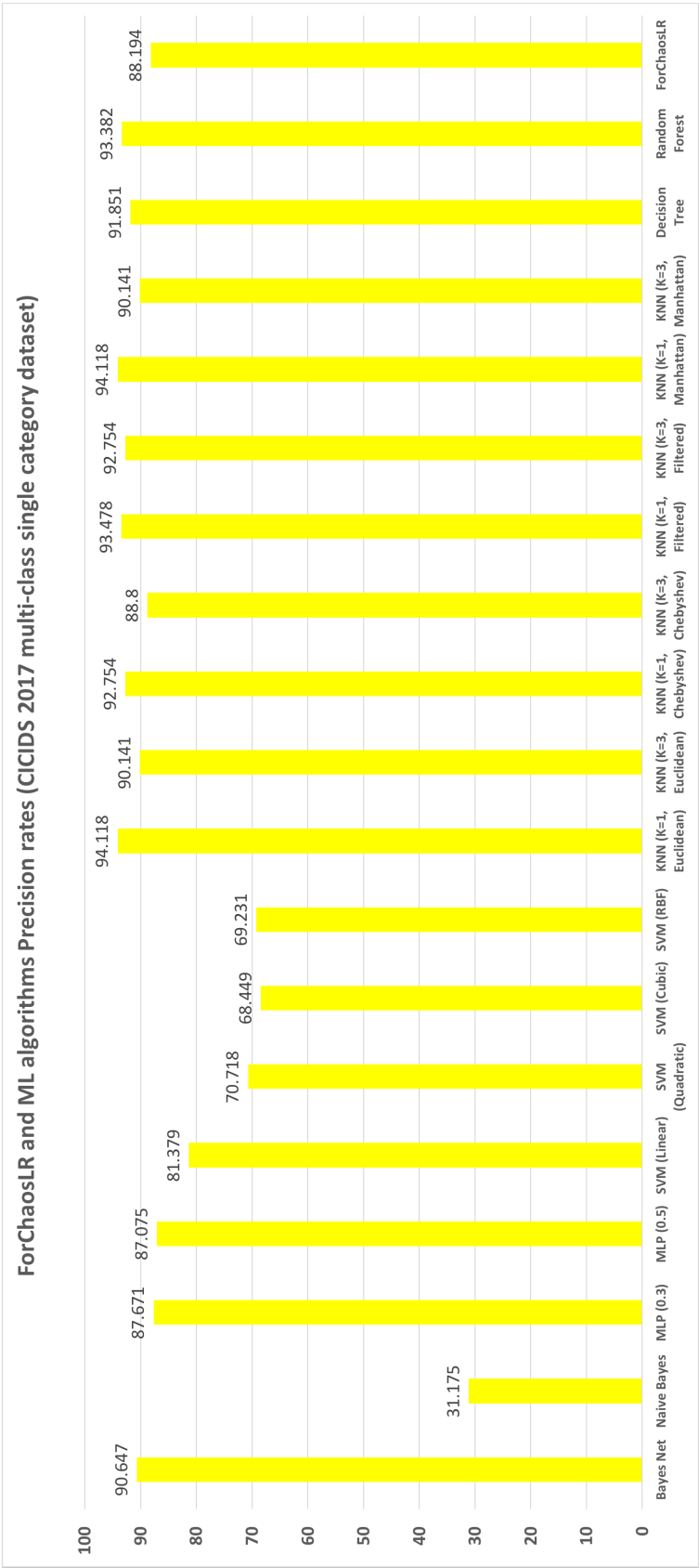


Fig. 8.9 ForChaosLR and ML algorithms Precision rates for CICIDS2017 single category multi-class dataset

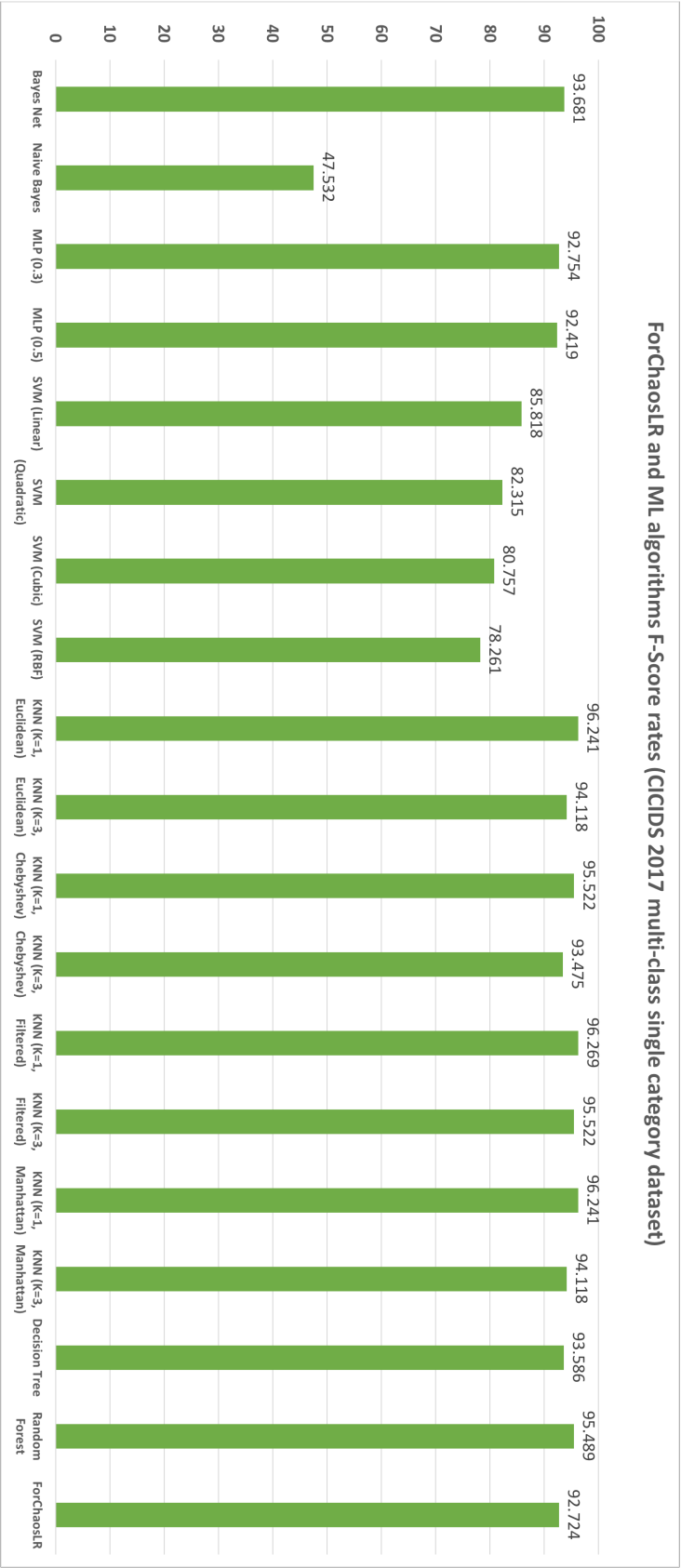


Fig. 8.10 ForChaosLR and ML algorithms F-Score rates for CICIDS2017 single category multi-class dataset

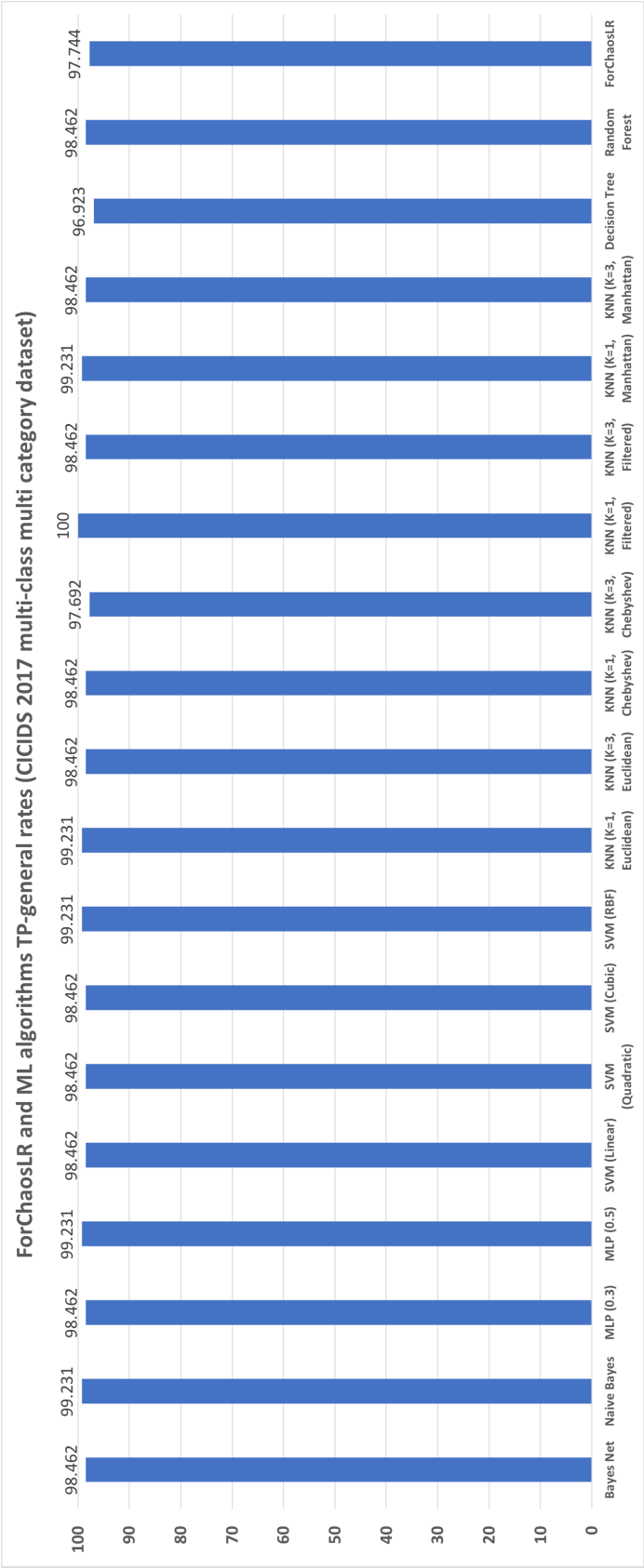


Fig. 8.11 ForChaosLR and ML algorithms TP rates for CICIDS2017 multi category multi-class dataset

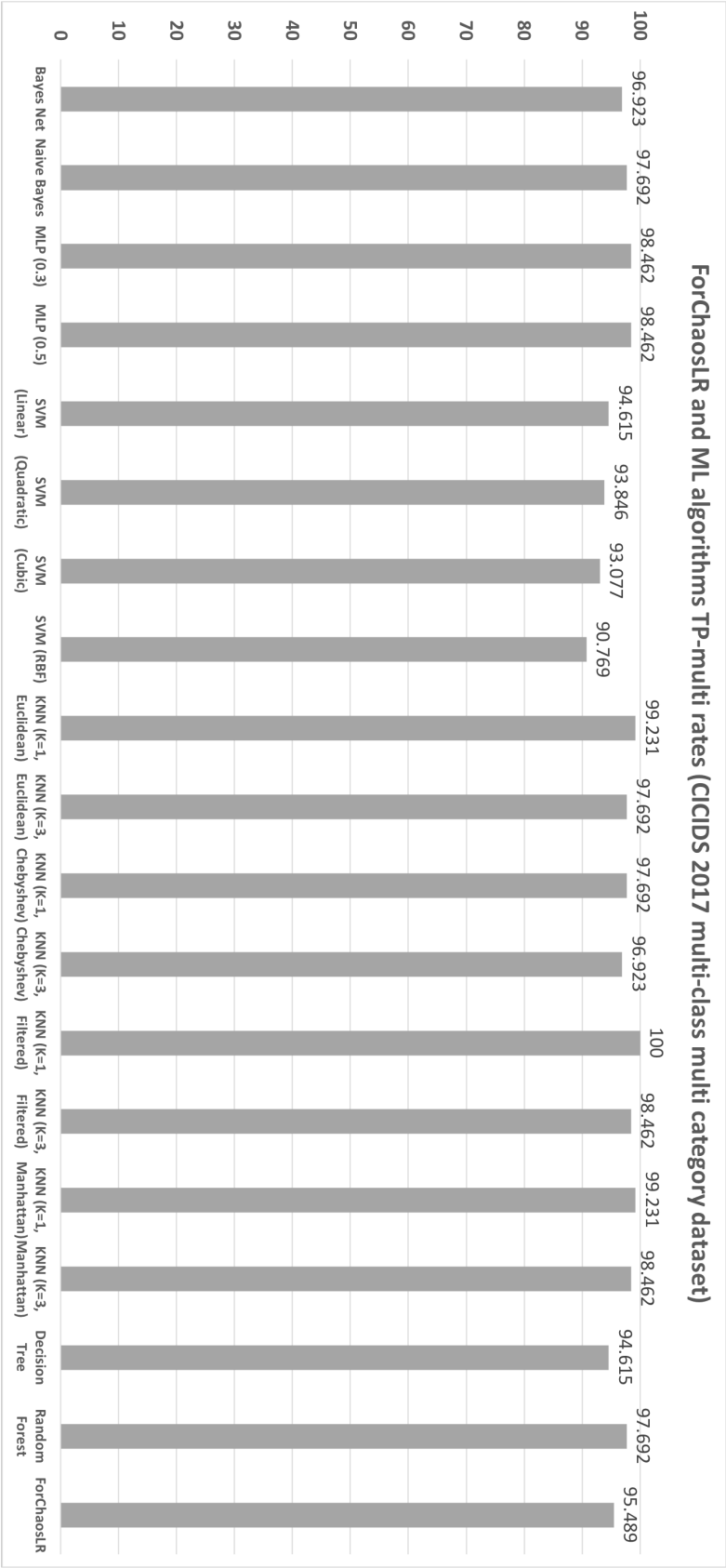


Fig. 8.12 ForChaosLR and ML algorithms TP-multi for CICIDS2017 multi category multi-class dataset

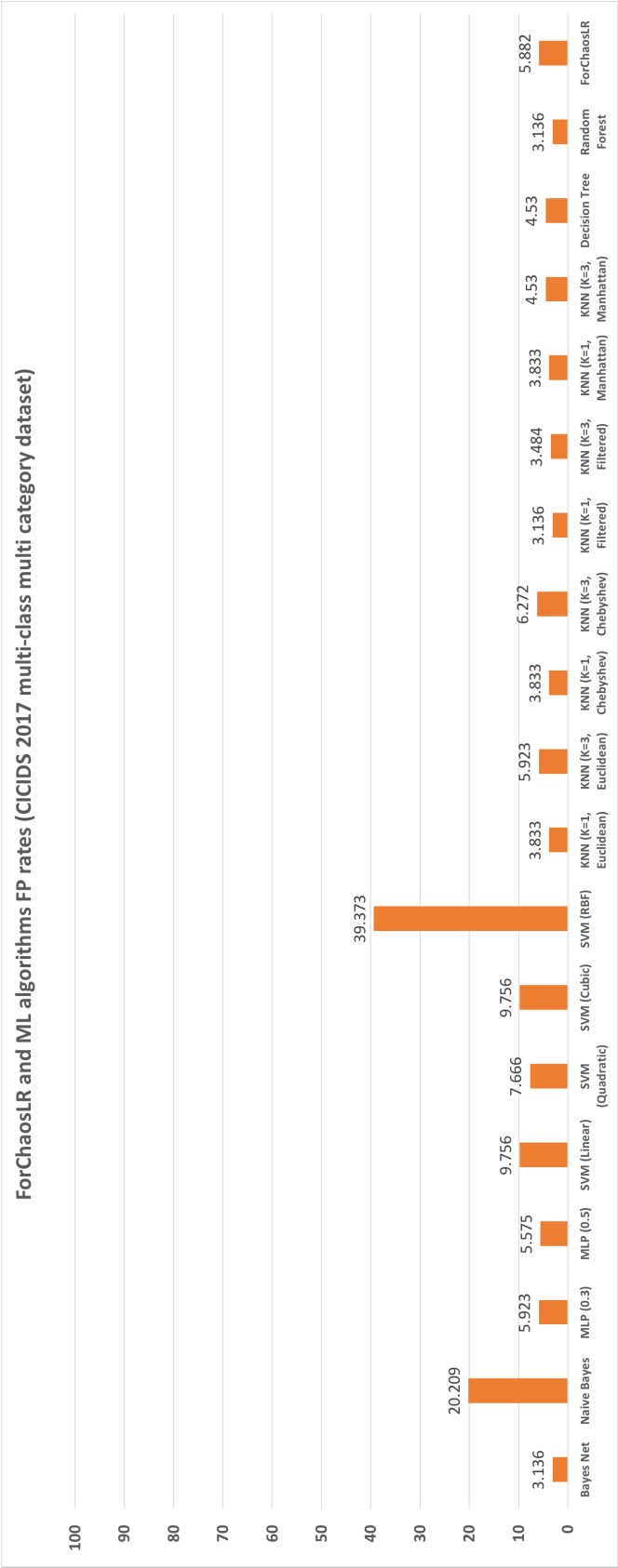


Fig. 8.13 ForChaosLR and ML algorithms FP rates for CICIDS2017 multi category multi-class dataset

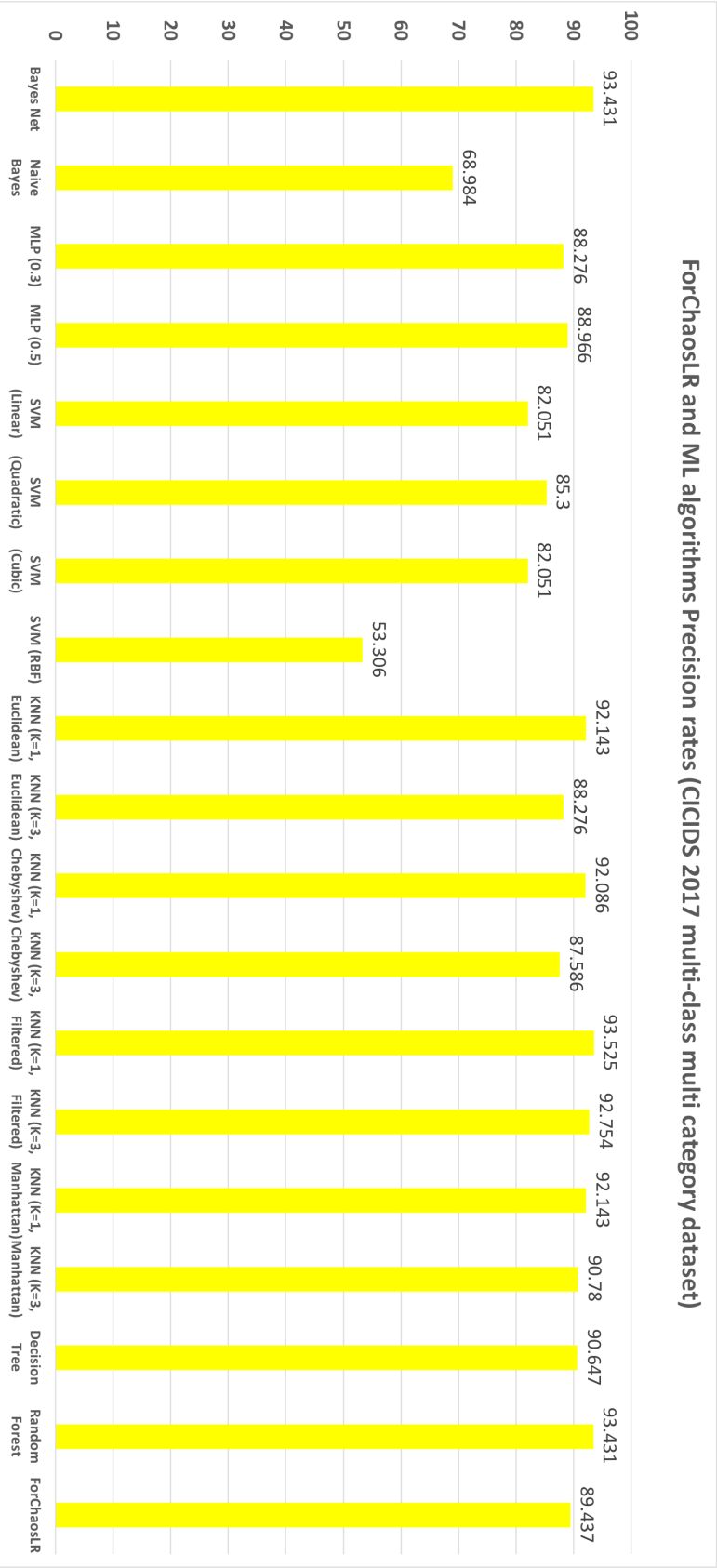


Fig. 8.14 ForChaosLR and ML algorithms Precision rates for CICIDS2017 multi category multi-class dataset

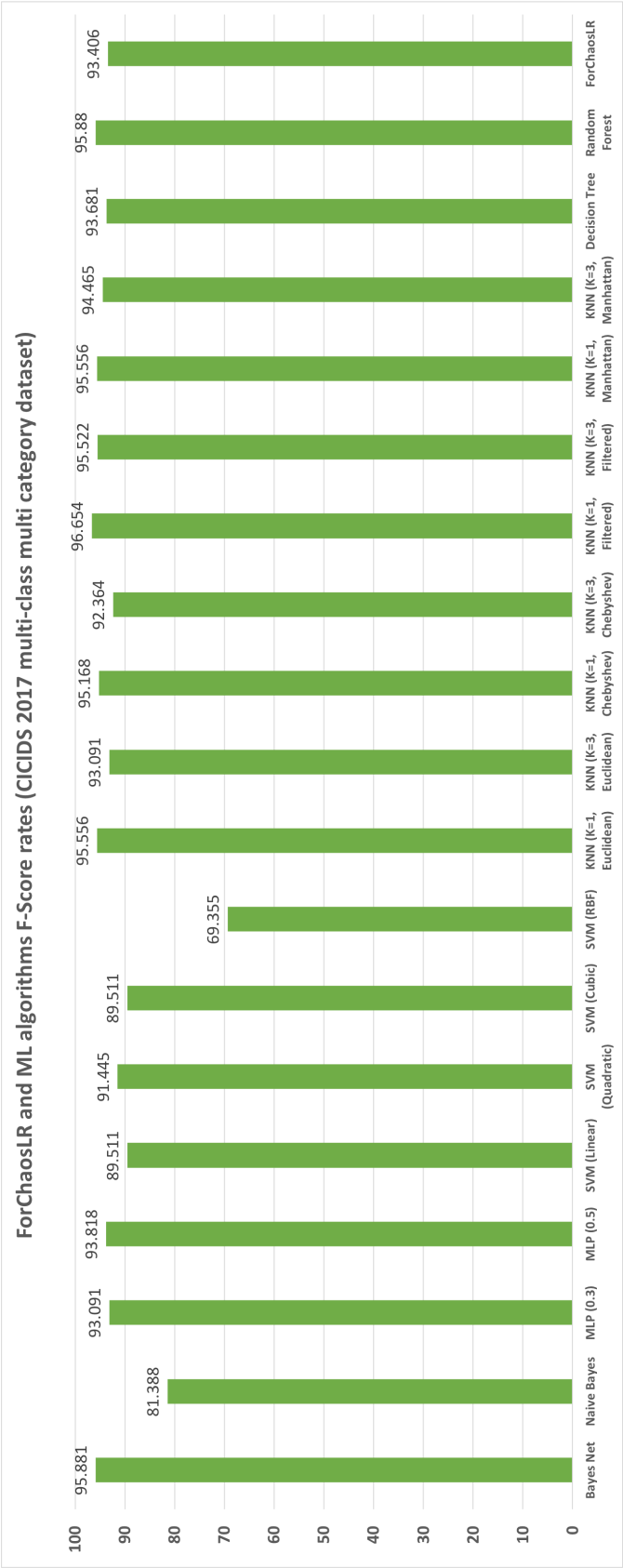


Fig. 8.15 ForChaosLR and ML algorithms F-Score rates for CICIDS2017 multi category multi-class dataset

Chapter 9

ForChaosLR evaluation in Raspberry Pi 3 model B+

9.1 Overview

ForChaosLR multi-layer IDS is primarily proposed for low-resource constrained appliances and IoT devices present in the smart home-smart grid network. Therefore, it is a necessity to evaluate its performance on such a device. As part of the objectives outlined in Chapter 1, it is essential that our proposed system:

- Should not occupy considerable amount of memory of a resource-constrained device.
- Should not considerably increase the CPU power of a resource-constrained device.
- Should not require lengthy amount of time for training and testing procedures.

In general, it is important to test a software application regarding its memory and CPU requirements regardless of the capabilities of the target computing machine. In addition, since ForChaosLR multi-layer IDS consists of a training phase it is important to also measure the time needed for constructing the algorithmic model as well as the testset evaluation time. Unarguably, since a low resource constrained device has limited capabilities in memory and CPU power, the performance of algorithmic tasks is going to be more time-consuming. Hence, we considered the following parameters:

- Memory usage which is the size of memory needed to execute a specific part of the application in megabytes, in our case training and testing code
- CPU power which is the percentage of CPU used for executing a specific part of the application.

- Training/Testing Time which is the time needed to execute and finish a specific part of the application.

To evaluate the requirements stated above, we deployed ForChaosLR multi-layer IDS's on a Raspberry Pi 3 model B+ using the datasets we already used in Chapter 6, 7 and 8 and described in Chapter 5, Sections 5.2.1, 5.2.2, and 5.2.3 so we could evaluate its hardware performance, specifically memory usage, CPU usage and training/testing time. In this chapter we discuss the results from these evaluations. To evaluate Raspberry Pi 3 model B+ device's resources requirements of ForChaosLR multi-layer IDS in further, we used the same set of ML algorithms used in the previous chapters, presented in Table 6.19. Weka library [144] was used for the implementation of all the algorithms.

9.2 Results Analysis and Discussion for Binary Datasets

9.2.1 Results Analysis and Discussion for NS-3 Datasets

In Tables 9.1 and Table 9.2 we present the results for the NS-3 binary dataset for the training and testing phases for both ForChaosLR multi-layer IDS and the set of ML algorithms. We summarise our general findings below:

- For the training phase, none of the ML algorithms require more than approximately 2.7 megabytes of RAM to construct their model and no more than approximately 5.9% of CPU.
- For the testing phase, none of the ML algorithms require more than approximately 1.9 megabytes of RAM and no more than 5.7% of CPU to evaluate the testing set provided.
- Regarding the training time needed we observe that the more complex algorithmic models correctly need more time to be constructed where the maximum amount is generated by SVM(RBF) 227 seconds which is approximately 3.78 minutes. Also, both neural network models (MLP(0.3) and MLP(0.5)) require 92-93 seconds for training.
- Regarding the testing time needed we observe that the longest time is needed by the KNN algorithms with the maximum testing time being approximately 2680 seconds. This was expected since KNN algorithms do not consist of an actual training phase. Instead, they perform "on the fly" training while classifying incoming instances. Hence, the big difference in time needed to evaluate the testing set.

Table 9.1 NS-3 Binary Training Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	2.698	4.924	3.382
Naive Bayes	2.330	3.717	2.436
MLP(0.3)	2.427	4.357	92.077
MLP(0.5)	2.427	5.471	93.305
SVM(Linear)	2.300	3.944	3.437
SVM(Quadratic)	2.301	4.905	17.724
SVM(Cubic)	2.301	4.857	29.061
SVM (RBF)	2.304	3.593	226.591
KNN(1, Euclidean)	2.338	5.083	2.437
KNN(3, Euclidean)	2.338	4.490	2.370
KNN(1, Chebyshev)	2.340	4.296	2.347
KNN(3, Cheyshev.)	2.340	4.743	2.376
KNN(1, Filtered)	2.555	3.897	2.884
KNN(3, Filtered)	2.555	4.197	2.821
KNN(1, Manhattan)	2.340	4.385	2.397
KNN(3, Manhattan)	2.340	5.876	2.422
Decision Tree	2.397	4.068	2.985
Random Forest	2.461	4.412	11.781
ForChaosLR	3.328	9.350	4.108
ForChaos	0.905	4.712	0.013

Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the algorithmic approach they follow for both the training and the testing phases.

Bayesian Algorithms :

- The Bayesian networks algorithm requires more time, memory and CPU power for both its training and testing phases compared to the naive Bayesian algorithm.
- This was expected, as Bayesian networks is a more complex model than naive Bayesian and therefore requires more resources to construct its Bayesian acyclic graph in contrast to the latter which calculates the maximum probability.

Neural Networks Algorithms :

- During training, both algorithms require approximately the same memory resources but MLP(0.5) needs more CPU power and training time construct its model than MLP(0.3).

- This is due to a higher learning rate calculates higher rate of weight change and therefore requires more resources.
- However, the differences are not significant since the memory usage is nearly equivalent and the CPU power less than 1%.
- Both neural network algorithms' training time is significantly higher than the rest of the ML algorithms and ForChaosLR with approximately 92-93 seconds, an expected result as the optimal calculation of weights for a neural network is a lengthy process.
- By examining the results from the testing phase, we observe a significant decrease across all three metrics (memory usage, CPU power and training/testing time). This was expected as neural networks' training phase is a lengthy and demanding in resources (due to the high in complexity training procedure) but not their testing phase.
- During testinh, we observe MLP(0.5) being slightly higher in resources than MLP(0.3, however the differences are not significant.
- Comparing both models to the rest of the ML algorithms during testing phase, no major differences are detected.

SVM Algorithms :

- During training, all SVM algorithms require approximately the same memory and CPU power to construct their models however with no significant differences from each other.
- During testing, the same trend is observed however, SVM(RBF) requires more memory than the polynomial SVM algorithms. However, once again, these differences are not considered significant.
- Hence, we conclude that based on the training/testing sets provided all of the SVM algorithms do not require significant memory and CPU and could therefore be deployed in a low-resource device with the capabilities of Raspberry Pi 3 model B+ device.
- The most notable difference in hardware requirements is the training and testing time. Based on the results obtained, we observe SVM(RBF) requires significantly more time in both cases than the polynomial kernels.
- Specifically, in training phase SVM(RBF) needs at least 200 additional seconds to construct its model and in testing phase at least 23 seconds to evaluate the testing

set. This was expected as SVM(RBF) performs a significant amount of additional calculations compared to the polynomial kernels.

KNN Algorithms :

- During both training and testing phases, the memory usage is approximately the same across all KNN algorithms with no significant differences from each other. Therefore, we conclude that both K number and distance do not play a significant role during both training and testing using the NS-3 binary dataset.
- However, the most significant difference is in testing times. This is due to the KNN algorithms not consisting of an actual training phase. Instead, they perform "on the fly" training while classifying incoming instances.
- Hence, the big difference in time needed to evaluate the testing set. Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the family they belong to.
- The larger number of K directly increases the testing evaluation time, an expected result as the higher number of K's means more calculations and comparison must be made between the nearest training instances and the instance to be classified.
- KNN(Filtered) algorithms require significantly more time, with more than 2500 seconds which is approximately 42 minutes than the rest of the KNN algorithms.
- This is practically infeasible in low-resource constrained and IoT devices, since it is essential for fast classification of incoming traffic.
- Regarding the rest of the KNN algorithms, their testing time required is significantly less, 85-124 seconds.

Tree Algorithms :

- Random forest algorithm requires more resources in memory and CPU power than decision tree algorithm during both training and testing phases. This was expected as the former constructs multiple decision trees in contrast to the latter during training. During testing, the data passes from multiple trees and therefore more resources are needed. However, their differences cannot be considered as highly significant.
- The significant difference between random forest and decision tree is the training/testing time. The time difference is approximately 9 seconds for training and 2.5 seconds for testing for the same reasons presented above.

ForChaosLR results and comparison with ML algorithms :

- During training, ForChaosLR requires an additional 0.8-1 megabyte and 3.5-5.5% CPU power compared to the ML algorithms. However, these differences cannot be considered highly significant.
- This is possibly due to ForChaosLR essentially needing to train two distinct algorithms with different data; ForChaos first-layer needs a small set of legitimate real-time past observational instances while LR second-layer needs a large set of both malicious and legitimate offline instances.
- However, the training and testing times for ForChaosLR were not higher than the highest training and testing times of the ML algorithms. ForChaosLR requires less training time than more complex ML algorithms such as neural networks, SVM(Quadratic), SVM(Cubic) and SVM(RBF) and random forest and less evaluation time than ML algorithms such as KNN and SVM(RBF) algorithms.
- In contrast, ForChaos requires less memory and training/testing time than all the ML algorithms.
- This is due to ForChaos only requiring a very small amount of training size to construct its model, so the training time and memory are not high.
- On the other hand, ForChaos requires a considerable amount of CPU compared to a number of the ML algorithms during training. This is due to a considerable amount of calculations that should be done in order to construct the forecasting model, including constructing predictions for incoming traffic, calculate the error between forecasting and actual values and calculate the RMSE error. Based on the error achieved, different combinations for α and window sizes are evaluated. During evaluation, ForChaos requires less CPU than the ML algorithms due to it only calculating the forecasting value for future incoming traffic, then the error between them and the Lyapunov exponent.

9.2.2 Results Analysis and Discussion for ISCXIDS 2012 Dataset

In Tables 9.3 and 9.4 we present the results for the ISCXIDS 2012 binary dataset for the training and testing phases for both ForChaosLR multi-layer IDS and the set of ML algorithms. We summarise the general findings below:

Table 9.2 NS-3 Binary Test Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	1.497	4.193	2.408
Naive Bayes	1.313	1.892	2.313
MLP (0.3)	1.521	3.545	2.293
MLP (0.5)	1.804	3.974	2.363
SVM (Linear)	1.388	4.585	2.162
SVM (Quadratic)	1.125	4.791	5.024
SVM (Cubic)	1.125	5.431	6.985
SVM (RBF)	1.871	4.163	30.767
KNN (K=1, Euclidean)	1.022	4.052	94.791
KNN (K=3, Euclidean)	1.029	4.909	104.933
KNN (K=1, Chebyshev)	1.027	4.615	85.165
KNN (K=3, Chebyshev)	1.028	5.143	93.145
KNN (K=1, Filtered)	1.066	4.385	2646.416
KNN (K=3, Filtered)	1.066	5.097	2678.812
KNN (K=1, Manhattan)	1.009	4.007	111.452
KNN (K=3, Manhattan)	1.027	5.671	124.124
Decision Tree	1.334	4.227	1.932
Random Forest	1.402	4.524	4.380
ForChaosLR	1.444	6.176	3.758
ForChaos	0.015	0.993	0.551

- For the training phase, none of the ML algorithms require more than approximately 2.4 megabytes and 6.9% of CPU power.
- For the testing phase, none of the ML algorithms require more than approximately 1.9 megabytes and 6.6% of CPU power.
- With regards to training time, we once again observe both neural network models and SVM(RBF) requiring significantly more time to construct their models compared to the rest of the ML algorithms.
- This trend was also evident in the NS-3 binary training set results however, the numbers for the ISCXIDS 2012 training set were significantly lower. We assume this is due to this training set only having malicious flooding and normal instances. Hence, it is easier to construct a model where the difference between the two classes is clear and distinct.

Table 9.3 ISCXIDS 2012 Training Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	2.423	5.539	3.291
Naive Bayes	1.906	4.500	2.406
MLP (0.3)	1.970	4.825	82.875
MLP (0.5)	2.156	4.973	83.708
SVM (Linear)	2.303	4.327	3.003
SVM (Quadratic)	2.315	3.987	3.857
SVM (Cubic)	2.319	3.894	4.396
SVM (RBF)	2.314	4.637	29.988
KNN (K=1, Euclidean)	1.797	4.699	2.478
KNN (K=3, Euclidean)	1.984	6.860	2.414
KNN (K=1, Chebyshev)	1.986	4.573	2.636
KNN (K=3, Chebyshev)	1.986	4.926	2.391
KNN (K=1, Filtered)	2.325	3.997	2.772
KNN (K=3, Filtered)	2.325	5.230	2.806
KNN (K=1, Manhattan)	1.986	4.246	2.458
KNN (K=3, Manhattan)	1.986	6.014	2.367
Decision Tree	1.834	4.170	2.591
Random Forest	2.186	5.031	7.046
ForChaosLR	2.727	8.680	4.862
ForChaos	0.905	4.712	0.013

- With regards to testing time, we once again observe that the longest time is needed by the KNN algorithms with the maximum testing time approximately being 475 seconds.
- This trend was also evident in the NS-3 binary testing set results however, the numbers for the ISCXIDS 2012 testing set are significantly lower. This is due to the testing set simply being smaller in size.
- We also observe that all the ML algorithms, regardless of their complexity, required less memory/CPU power and execution time than their equivalent results in the NS-3 binary training and testing time results.

Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the family they belong to.

Bayesian Algorithms :

- Once again, Bayesian networks algorithm require more time, memory and CPU power for both its training and testing phases compared to the naive Bayesian algorithm.

- This was expected, as Bayesian networks is a more complex model than naive Bayesian and therefore requires more resources to construct its Bayesian acyclic graph in contrast to the latter which calculates the maximum probability.
- This trend was also present in the NS-3 binary training and testing set results however, the results do not show significant differences therefore, there is no indication that the absence of slow-rate instances during training promotes these algorithms to be less demanding in resources.

Neural Networks Algorithms :

- During both training and testing, both neural network models require approximately the same amount of memory and CPU power, with MLP(0.5) needing slightly more resources than MLP(0.3).
- Regarding training time needed, we once again observe that neural networks require a considerable amount of time to construct their models which is significantly higher than most ML algorithms.
- In contrast, during testing phase both neural network models require no more than 1.4 seconds to classify the test set. This was expected, as the evaluation time using a neural network is significantly less than its training in terms of time required.
- Similar trends were also evident on the NS-3 binary training and test set results. We should highlight that both training and test results were decreased when compared with the equivalent ones from NS-3 binary training and test set results across all metrics. We believe this is due to the absence of slow-rate instances in the training set. Therefore, the algorithmic model is likely to be less complex and will need less time to be constructed.

SVM Algorithms :

- During training and testing phases, all of the SVM algorithms require approximately the same amount of memory and CPU power with no significant differences from each other.
- With regards to training time, SVM(RBF) require a considerable amount of time compared to the rest of the SVM algorithms and the remaining ML algorithms while in testing phase the times generated by all SVM algorithms are approximately equivalent.

- When comparing this set of results to the NS-3 binary training and test set equivalent results, we observe that all of the SVM algorithms require significantly less time to execute the relevant task (construct the model using the training set or evaluate the test set).
- Specifically, this is evident with the SVM(Quadratic), SVM(Cubic) and SVM(RBF) during training where the difference is approximately 14, 25 and 200 seconds respectively.
- It is also evident during testing with SVM(Cubic) and SVM(RBF) where the difference is 5.5 and 28 seconds respectively.
- Hence, we observe that in higher degree or more complex kernels, the absence of slow-rate instances has a direct impact in training time as the final constructed model is likely to be less complex and therefore needs less time.

KNN Algorithms :

- During training and testing, all the KNN algorithms require approximately similar memory resources and CPU power regardless of K number and distance measure.
- During both training and testing, we don't observe any significant differences between the different KNN algorithms with regards to their memory usage and CPU power requirements.
- The current results do demonstrate significant differences from the NS-3 binary training and test set equivalent results. This indicates that the removal of slow-rate instances impacts the functioning of the KNN algorithms used with regards to their memory usage and CPU power requirements.
- With regards to testing time, we once again observe KNN algorithms require significantly more time during testing as opposed to the rest of the ML algorithms due to their "lazy training" approach. Furthermore, the testing time is slightly increased when the K number is increased.
- Compared to the NS-3 binary test set results, the respective test times are significantly lower but this is only due to the ISCXIDS 2012 test set is significantly lower in size and due to the absence of slow-rate instances is evident in the results.

Tree Algorithms :

- During training phase, we observe that random forest algorithm requires more memory, CPU power and training time than decision tree algorithm.
- Once again, this is because multiple trees must be constructed during training as opposed to decision tree algorithm where there is only one.
- The same trend is observed during testing, where random forest requires more memory, CPU power and testing time than decision tree algorithm.
- This occurs due to incoming traffic being classified by all the trees constructed during training and then performing majority voting.
- When comparing this set of results to the NS-3 binary training set results, we observe that during training the memory usage and training time are considerably lower. This highlights once again that the absence of slow-rate instances contributes to lower hardware requirements.

ForChaosLR results and comparison with ML algorithms :

- ForChaosLR demonstrates the same trend as the ML algorithms where it requires less memory, CPU power and time for both training and testing phase when compared to the NS-3 binary training and testing set results. Hence, we conclude that ForChaosLR is also affected by the absence of the slow-rate instances.
- In contrast to the more complex algorithms such as both neural networks models, random forest and SVM(RBF) ForChaosLR does not require a large amount of time to be trained despite consisting of two distinct algorithms.
- However, it requires more time during evaluation as well as more memory and CPU power during both training and testing compared to the ML algorithms.
- In contrast, ForChaos requires less memory and training/testing time than all the ML algorithms. Once again, ForChaos requires a considerable amount of CPU compared to a number of the ML algorithms during training but less during testing. This is once again due to the reasons already mentioned in Section 9.2.1.

9.2.3 Results Analysis and Discussion for CICIDS 2017 Dataset

In Tables 9.5 and 9.6 we present the results for the CICIDS 2017 binary dataset for the training and testing phases for both ForChaosLR multi-layer IDS and the set of ML algorithms. We summarise the most important findings below:

Table 9.4 ISCXIDS 2012 Test Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	1.545	4.333	1.347
Naive Bayes	1.271	3.711	1.302
MLP (0.3)	1.796	3.819	1.355
MLP (0.5)	1.796	4.444	1.376
SVM (Linear)	1.660	3.345	1.442
SVM (Quadratic)	1.660	4.437	1.592
SVM (Cubic)	1.877	5.525	1.604
SVM (RBF)	1.660	4.756	2.704
KNN (K=1, Euclidean)	1.699	5.228	27.840
KNN (K=3, Euclidean)	1.699	5.342	29.027
KNN (K=1, Chebyshev)	1.647	4.506	25.50
KNN (K=3, Chebyshev)	1.646	5.137	27.172
KNN (K=1, Filtered)	1.283	3.709	472.710
KNN (K=3, Filtered)	1.283	4.928	475.319
KNN (K=1, Manhattan)	1.561	5.582	29.472
KNN (K=3, Manhattan)	1.561	5.673	30.694
Decision Tree	1.527	4.797	1.187
Random Forest	1.661	6.642	1.552
ForChaosLR	1.780	4.443	3.040
ForChaos	0.377	0.243	1.330

- For the training phase, none of the ML algorithms require more than approximately 2.6 megabytes and 5.9% of CPU power.
- For the testing phase, none of the ML algorithms require more than approximately 1.7 megabytes and 5.9% of CPU power.
- Regarding training time, we observe SVM(Cubic), SVM(RBF) and both neural network models require more time than the rest of the ML algorithms. Although the training set consists of slow-rate instances it is significantly smaller in size than both ISCXIDS 2012 and NS-3 binary training sets. Therefore, we conclude that the training set size can play a significant role in training time for these algorithms.
- Regarding testing time, we only observe the KNN(Filtered) algorithms taking a considerable lengthier amount of time than the rest of the ML algorithms, including the rest of the KNN algorithms. This indicates to us that KNN algorithms' testing performance is affected by the testing set size.

Table 9.5 CICIDS 2017 Training Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	2.635	5.870	2.717
Naive Bayes	2.237	4.065	2.164
MLP (0.3)	2.385	4.473	40.769
MLP (0.5)	2.385	5.036	41.253
SVM (Linear)	2.501	4.333	2.924
SVM (Quadratic)	2.519	4.333	14.136
SVM (Cubic)	2.519	4.483	20.887
SVM (RBF)	2.519	4.394	64.737
KNN (K=1, Euclidean)	2.221	5.001	2.217
KNN (K=3, Euclidean)	2.221	5.321	2.315
KNN (K=1, Chebyshev)	2.226	4.931	2.254
KNN (K=3, Chebyshev)	2.226	3.876	2.226
KNN (K=1, Filtered)	2.483	4.597	2.497
KNN (K=3, Filtered)	2.483	4.383	2.459
KNN (K=1, Manhattan)	2.226	4.807	2.153
KNN (K=3, Manhattan)	2.226	4.665	2.237
Decision Tree	2.230	4.650	2.565
Random Forest	2.438	5.217	6.625
ForChaosLR	3.244	9.095	4.941
ForChaos	0.905	4.712	0.013

- Comparing this set of results with the ISCXIDS 2012 respective ones, we observe that the memory usage is slightly higher. This indicates that even though the CICIDS 2017 training set is smaller in size than the ISCXIDS 2012 training set, the inclusion of slow-rate instances forces all of the ML algorithms to perform additional calculations so they can construct a more complex algorithmic model which needs more memory.

Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the family they belong to.

Bayesian Algorithms :

- During both training and testing phases, Bayesian networks algorithm requires more memory, CPU power and training/testing time than naive Bayesian algorithm.
- This was expected, as Bayesian networks is a more complex model than naive Bayesian and therefore requires more resources to construct its Bayesian acyclic graph in contrast to the latter which calculates the maximum probability.

- The same trend was present in both the NS-3 binary and ISCXIDS 2012 training and testing sets.
- Hence, we conclude that regardless of the introduction of slow-rate instances and/or training/testing set size Bayesian networks algorithm is likely to require more resources than naive Bayesian algorithm for binary classification.

Neural Networks Algorithms :

- During both training and testing, both neural network models require approximately the same amount of memory and CPU power.
- Regarding training time, both MLP neural network algorithms need the highest amount of training time compared to the rest of the ML algorithms, except SVM(RBF).
- This indicates that regardless of slow-rate instances and/or training set size MLP neural network algorithms need a considerable amount of time to construct their model in contrast to the rest of the ML algorithms.
- However, we have to highlight that the current training time is considerably lower than the NS-3 binary and ISCXIDS 2012 equivalent results. This indicates that the training set size has a direct impact on training time.
- In contrast, during testing phase both neural network models need no more than 1.1 seconds to classify the test set.

SVM Algorithms :

- During training and testing phases, all of the SVM algorithms require approximately the same amount of memory and CPU power with no significant differences from each other, a trend which was also present in both NS-3 binary results and ISCXIDS 2012 dataset results. This demonstrates that regardless of the higher complexity between the different SVM algorithms, their memory usage and CPU power required for training is similar.
- Comparing each SVM algorithm's memory usage and CPU power with its NS-3 binary and ISCXIDS 2012 datasets equivalent results we observe that all of them require slightly increased resources. Although this increase cannot be considered significant, we conclude that the introduction of slow-rate instances in training increases the required resources, regardless of the training set being smaller in size.

- Regarding training time, we observe that except SVM(Linear), the rest of the SVM algorithms need a significant time to construct their algorithmic model compared to the majority of ML algorithms used. Specifically, SVM(RBF) needs the most training time from all of the ML algorithms while in the NS-3 binary and ISCXIDS 2012 datasets results both neural network MLP algorithms needed the longest training time. In fact, SVM(RBF) requires the longest time for the NS-3 binary dataset. This highlights that SVM(RBF) is directly affected by the introduction of slow-rate instances in the attack class and needs more time to effectively construct an accurate algorithmic model.
- Furthermore, comparing the training time results with the equivalent ones from the NS-3 binary dataset, we observe that in this series of experiments SVM(Quadratic) and SVM(Cubic) need significantly more training time than when the former two training sets were used. This also confirms and justifies our conclusions stated in the previous point.
- During testing, there are no significant differences with regards to memory usage and CPU power. Also, when compared to the equivalent results from the NS-3 binary and ISCXIDS 2012 datasets, there are no significant differences. This highlights that regardless of the testing size the SVM algorithms are unaffected by it.
- However, SVM(RBF) requires more testing time in NS-3 binary dataset. This highlights that it requires more time to classify traffic due to the model's complexity.

KNN Algorithms :

- During both training and testing, all of the KNN algorithms once again need approximately similar memory resources and CPU power regardless of K number and distance measure.
- With regards to training results, the memory usage and CPU power results obtained in this series of training experiments are slightly increased compared to the equivalent ones from the ISCXIDS 2012 dataset. This indicates that the introduction of slow-rate instances affects the KNN algorithms' resources requirements. This finding is also evident in the NS-3 binary results.
- In contrast, the memory usage and CPU power results obtained in this series of testing experiments are slightly decreased compared to the equivalent ones from the ISCXIDS 2012 dataset. This indicates that the testing size directly affects the resources as expected. This occurs due to the lazy approach KNN algorithms take where they perform training on the fly while classifying new incoming traffic.

- With regards to training time, we do not observe any differences between this series of results and the ISCXIDS 2012 and NS-3 binary equivalent results for any of the KNN algorithms.
- With regards to testing time only KNN(Filtered) needs significantly more time than the rest of the ML algorithms. Furthermore, in this series of experiments only KNN(Filtered) needs more testing time as opposed to the ISCXIDS 2012 and NS-3 binary results where all of the KNN algorithms needed significantly more testing time than the rest of the ML algorithms. This indicates that regardless of the introduction of slow-rate instances, the testing set size plays a significant role in testing time.

Tree Algorithms :

- During training phase, we observe that random forest algorithm requires more memory, CPU power and training time than decision tree algorithm. Once again, the reason for this set of results is because multiple trees must be constructed during training as opposed to decision tree algorithm where there is only one. However, these differences cannot be considered highly significant.
- The same trend is observed during testing, where random forest needs more memory, CPU power and testing time than decision tree algorithm. This occurs due to incoming traffic being classified by all the trees constructed during training and then performing majority voting.
- Comparing the training results to the equivalent ones from the ISCXIDS 2012 and NS-3 binary datasets, we observe that the current results require more resources (memory usage and CPU power) than the ISCXIDS 2012 ones and are similar to the NS-3 binary datasets. This indicates that training size and the introduction of slow-rate instances has a direct impact on the KNN algorithms memory, CPU power and training/testing time requirements.
- During training, random forest needs significantly more time to construct its model compared to the decision tree algorithm. However, during testing their testing times are approximately equivalent.

ForChaosLR results and comparison with ML algorithms :

- During training and testing, ForChaosLR requires more memory and CPU power but approximately the same training time than the ISCXIDS 2012 equivalent results. This indicates that the introduction of slow-rate instances directly impact the hardware requirements.

- In addition, ForChaosLR requires more memory and CPU power than all ML algorithms during both training and testing phases.
- In contrast to the more complex algorithms such as both neural networks MLP models, SVM(Quadratic), SVM(Cubic) and SVM(RBF) ForChaosLR does not require a large amount of time to be trained despite consisting of two distinct algorithms. In addition, it requires less testing time than all KNN algorithms as well as all SVM algorithms (except SVM (Linear)).
- In contrast, ForChaos requires less memory and training/testing time than all the ML algorithms. Once again, ForChaos requires a considerable amount of CPU compared to a number of the ML algorithms during training but less during testing. This is once again due to the reasons already mentioned in Section 9.2.1.

Table 9.6 CICIDS 2017 Test Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	1.635	4.223	1.174
Naive Bayes	1.288	3.918	1.026
MLP (0.3)	1.603	4.392	1.111
MLP (0.5)	1.603	4.633	1.116
SVM (Linear)	1.670	4.387	1.110
SVM (Quadratic)	1.671	4.511	1.521
SVM (Cubic)	1.690	4.885	1.751
SVM (RBF)	1.694	5.886	2.252
KNN (K=1, Euclidean)	1.467	5.255	2.395
KNN (K=3, Euclidean)	1.467	5.221	2.578
KNN (K=1, Chebyshev)	1.464	4.433	2.344
KNN (K=3, Chebyshev)	1.464	2.726	2.488
KNN (K=1, Filtered)	1.423	4.520	39.088
KNN (K=3, Filtered)	1.423	5.834	39.195
KNN (K=1, Manhattan)	1.464	4.741	2.583
KNN (K=3, Manhattan)	1.464	4.174	2.734
Decision Tree	1.429	4.191	1.001
Random Forest	1.474	5.501	1.207
ForChaosLR	1.993	7.220	1.165
ForChaos	0.429	3.423	0.089

9.3 Results Analysis and Discussion for Multi-Class Datasets

9.3.1 Results Analysis and Discussion for NS-3 Multi-Class Dataset

Table 9.7 NS-3 Multi-Class Training Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	2.522	5.374	3.175
Naive Bayes	2.265	4.844	2.419
MLP (0.3)	2.268	5.129	63.011
MLP (0.5)	2.296	5.339	63.541
SVM (Linear)	1.517	4.208	3.136
SVM (Quadratic)	1.417	4.005	5.387
SVM (Cubic)	1.415	4.081	6.381
SVM (RBF)	1.414	4.438	53.423
KNN (K=1, Euclidean)	2.279	5.218	2.354
KNN (K=3, Euclidean)	2.279	5.229	2.485
KNN (K=1, Chebyshev)	2.280	4.789	2.340
KNN (K=3, Chebyshev)	2.280	4.827	2.382
KNN (K=1, Filtered)	2.308	4.883	2.749
KNN (K=3, Filtered)	2.308	4.508	2.691
KNN (K=1, Manhattan)	2.280	5.479	2.388
KNN (K=3, Manhattan)	2.280	5.774	2.379
Decision Tree	2.314	4.027	2.780
Random Forest	2.432	4.655	8.086
ForChaosLR	3.262	9.473	9.504
ForChaos	0.905	4.712	0.013

In Tables 9.7 and 9.8 we present the results for the NS-3 multi-class datasets for the training and testing phases for both ForChaosLR multi-layer IDS and the set of ML algorithms. We summarise the most important findings below:

- For the training phase, none of the ML algorithms require more than approximately 2.5 megabytes and 5.8% of CPU power.
- For the testing phase, none of the ML algorithms require more than approximately 1.9 megabytes and 6.6% of CPU power.

- When compared to the equivalent results from the NS-3 binary results we observe that there are no significant differences to memory usage and CPU power across all ML algorithms.
- Regarding training time, we observe SVM(RBF) and both neural network MLP models taking a considerable longer time than the rest of the ML algorithms. In contrast to the NS3-binary results, we observe that both MLP algorithms, SVM(Quadratic), SVM(Cubic) and SVM(RBF) requires additional training time. This indicates that with a multi-class training approach is a lengthier process with regards to the model construction for the algorithms mentioned.
- Regarding testing time, we observe the KNN algorithms once again require the longest time to classify incoming traffic for the reasons already stated. However, in this series of experiments KNN algorithms require significantly less time than their results in the NS-3 binary results. This indicates that a multi-class approach reduces the testing time for KNN algorithms.

Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the algorithmic category they belong to.

Bayesian Algorithms :

- During training and testing phase, with regards to memory usage, CPU power and training/testing time, Bayesian networks algorithm has higher requirements than naive Bayesian algorithm.
- This was expected, as Bayesian networks is a more complex model than naive Bayesian and therefore requires more resources to construct its Bayesian acyclic graph in contrast to the latter which calculates the maximum probability.
- When compared to the NS-3 binary equivalent results, we also observe the same trend of results with no significant differences between each metric.
- This indicates that the multi-class approach followed does not seem to have an impact on the hardware requirements of both Bayesian algorithms.

Neural Networks Algorithms :

- During both training and testing, both neural network models require approximately the same amount of memory and CPU power and training time, with MLP(0.5) needing slightly more resources than MLP(0.3). This finding was also present in the NS-3

binary results and we therefore conclude that a multi-class training approach does not affect the training of neural networks.

- When compared to the NS-3 binary equivalent results we do not observe any significant differences with regards to memory usage and CPU power.
- Regarding training time, both MLP neural network algorithms require the highest amount of training time compared to the rest of the ML algorithms. When compared to the equivalent results from the NS-3 binary training set we observe that a multi-class training approach substantially reduces training time. Hence, a separation of the DDoS attacks in specific types is beneficial to the algorithmic construction.
- Regarding testing results, we observe that there are no significant differences in the required resources compared to the NS-3 binary dataset equivalent results.

SVM Algorithms :

- During training and testing, all SVM algorithms require approximately the same memory and CPU power with their differences not being significant.
- When compared to the NS-3 binary results, we observe that the former results are decreased in memory usage and CPU power. This indicates that a multi-class training approach actually benefits the SVM algorithms and they require less memory and CPU power to construct their model.
- Regarding training and testing times, we observe that SVM(RBF) needs significantly more time than the rest of the SVM algorithms. When compared to the equivalent results from the NS-3 binary dataset we draw the following conclusions:
 - In the NS-3 binary dataset results, SVM(Quadratic), SVM(Cubic) and SVM(RBF) require significantly more time than the majority of the ML algorithms. This indicates that a multi-class approach benefits the SVM algorithms in training time required.
 - Furthermore, even though SVM(RBF) requires more training time than the rest of the SVM algorithms, its training time is significantly less than the SVM(RBF) for the NS-3 binary training set, with an approximate difference of 170 seconds.
 - In contrast, when comparing the two testing times of SVM(RBF), it requires approximately 8 additional seconds when the NS-3 multi-class dataset is used. However, this difference cannot be considered significant.

KNN Algorithms :

- During training, the memory usage and CPU power requirements and training time are similar by all KNN algorithms. Hence, we conclude that the K number and the distance metric do not play a significant role in the hardware requirements.
- When compared to the equivalent results from the NS-3 binary dataset we do not observe any significant differences. Therefore, a multi-class approach does not affect in any way the training set results.
- During testing, the memory usage and CPU power requirements are similar by all KNN algorithms. Hence, we conclude that the K number and the distance metric do not play a significant role in the hardware requirements.
- The most significant difference is in testing time. This is due to KNN algorithms not consisting of an actual training phase. Instead, they perform "on the fly" training while classifying incoming instances. Hence, the big difference in time needed to evaluate the testing set. Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the family they belong to.
- The larger number of K directly increases the testing evaluation time, an expected result as the higher number of K's means more calculations and comparison must be made between the nearest training instances and the instance to be classified.
- KNN(Filtered) algorithms require a significantly more time, with more than 1480 (approximately 24.7 minutes).
- This is practically infeasible in low-resource constrained and IoT devices as it is essential for fast classification of incoming traffic.
- Regarding the rest of the KNN algorithms the testing phase time needed is significantly less, ranging from 54-77 seconds.
- These trends were also evident in the NS-3 binary testing results from all the KNN algorithms. When compared to the equivalent results from NS-3 binary dataset, we observe that the former results are significantly decreased to the latter ones. This shows that a multi-class approach significantly decreases the testing time of the KNN algorithms.
- However, the testing time from the KNN algorithms is still significantly longer than the rest of the ML algorithms.

Tree Algorithms :

- During training and testing phases, random forest requires more memory and CPU power than decision tree algorithm, a trend which was also evident in the NS-3 binary training set results. However, their differences cannot be considered highly significant.
- Comparing the training/testing times with the equivalent ones from the NS-3 binary dataset results we observe that random forest's training time is decreased by approximately by 3 seconds. Therefore, we conclude that the multi-class training benefits random forest in its model construction.

ForChaosLR results and comparison with ML algorithms :

- During training, ForChaosLR requires approximately the same amount of memory and CPU power with our previous experimental results obtained. In contrast, ForChaosLR requires approximately double the amount of training time with previous experimental results obtained.
- ForChaosLR requires an additional 0.75-1.8 megabyte and 4-5.5% CPU power compared to the ML algorithms.
- We believe this is due to ForChaosLR consisting of two distinct algorithms with different datasets used during training; ForChaos first-layer needs a small set of legitimate real-time past observational instances while LR second-layer needs a large set of both malicious and legitimate offline instances.
- However, the training and testing times for ForChaosLR were not higher than the highest training and testing times of the ML algorithms. Therefore, ForChaosLR needs less training time than more complex ML algorithms such as both MLP neural networks and SVM(RBF) and less testing time than more complex ML algorithms such as KNN and SVM(RBF) algorithms.
- In contrast, ForChaos requires less memory and training/testing time than all the ML algorithms. Once again, ForChaos requires a considerable amount of CPU compared to a number of the ML algorithms during training but less during testing. This is once again due to the reasons already mentioned in Section 9.2.1.

Table 9.8 NS-3 Multi-Class Test Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	1.891	3.617	2.278
Naive Bayes	1.021	3.755	2.589
MLP (0.3)	1.362	6.580	2.260
MLP (0.5)	1.349	4.662	2.232
SVM (Linear)	1.169	5.283	2.089
SVM (Quadratic)	1.470	4.189	3.809
SVM (Cubic)	1.470	3.983	4.865
SVM (RBF)	1.505	4.733	38.607
KNN (K=1, Euclidean)	1.037	3.786	61.967
KNN (K=3, Euclidean)	1.059	4.967	66.546
KNN (K=1, Chebyshev)	1.050	3.890	54.429
KNN (K=3, Chebyshev)	1.057	4.983	59.346
KNN (K=1, Filtered)	1.357	4.808	1483.419
KNN (K=3, Filtered)	1.345	4.924	1519.391
KNN (K=1, Manhattan)	1.039	3.941	70.350
KNN (K=3, Manhattan)	1.057	5.122	77.564
Decision Tree	1.254	4.370	1.943
Random Forest	1.412	5.332	4.397
ForChaosLR	1.959	5.782	10.351
ForChaos	0.015	0.993	0.551

9.3.2 Results Analysis and Discussion for CICIDS 2017 Single Category Multi-Class Dataset

In Tables 9.9 and 9.10 we present the results for CICIDS 2017 multi-class single category datasets for the training and testing phases for both ForChaosLR multi-layer IDS and the set of ML algorithms.

- For the training phase, none of the ML algorithms require more than approximately 2.6 megabytes and 6.2% of CPU power.
- For the testing phase, none of the ML algorithms require more than approximately 1.7 megabytes and 5.7% of CPU power.
- Regarding training time, we observe SVM(RBF) and both neural network MLP models taking a considerable longer time than the rest of the ML algorithms. When compared to the CICIDS 2017 binary training results we observe that in this series of experiments

Table 9.9 CICIDS 2017 Multi-Class Single Category Training Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (Seconds)</i>
Bayes Net	2.593	5.733	2.694
Naive Bayes	2.213	5.163	2.194
MLP (0.3)	2.277	4.945	36.943
MLP (0.5)	2.277	5.163	36.983
SVM (Linear)	2.041	3.936	3.469
SVM (Quadratic)	1.416	5.444	8.435
SVM (Cubic)	1.415	4.692	9.623
SVM (RBF)	1.352	4.858	90.739
KNN (K=1, Euclidean)	2.236	3.408	2.172
KNN (K=3, Euclidean)	2.236	5.435	2.151
KNN (K=1, Chebyshev)	2.224	4.849	2.207
KNN (K=3, Chebyshev)	2.224	4.201	2.176
KNN (K=1, Filtered)	2.401	3.477	2.435
KNN (K=3, Filtered)	2.401	5.695	2.388
KNN (K=1, Manhattan)	2.224	6.232	2.240
KNN (K=3, Manhattan)	2.224	4.390	2.126
Decision Tree	2.365	4.374	2.465
Random Forest	2.449	5.043	7.060
ForChaosLR	3.189	9.937	4.962

SVM(RBF) needs significantly more time but neural network MLP algorithms need less time. This indicates that a multi-class training approach affects all algorithms in a different way with regards to their construction time.

- Regarding testing time, we observe only KNN(Filtered) algorithms once again required the longest time to classify incoming traffic for the reasons already stated previously.
- The rest of the KNN algorithms however need significantly less testing time. This trend has also been evident in the CICIDS 2017 binary results.

Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the family they belong to.

Bayesian Algorithms :

- During both training and testing phases, Bayesian networks algorithm requires more memory, CPU power and training/testing time than naive Bayesian algorithm.

- These trends were evident in the CICIDS 2017 binary dataset results. Therefore, the training and testing phases of both algorithms are unaffected by a multi-class approach.

Neural Networks Algorithms :

- During both training and testing, both neural network models require approximately the same amount of memory and CPU power, with MLP(0.5) needing slightly more resources than MLP(0.3).
- The same trends were also observed in the CICIDS 2017 binary dataset results. This indicates that following a multi-class training approach does not change how neural network MLP algorithms behave when compared with each other.
- Regarding training time, both MLP neural network algorithms need the highest amount of training time compared to the rest of the ML algorithms, except SVM(RBF).
- We have to point out that in this round of experiments the training time was slightly decreased compared to the CICIDS 2017 binary dataset results. Hence, we conclude that a multi-class training approach assists MLP neural networks positively impacts the training time.
- On the other hand, testing time between the CICIDS 2017 multi-class and binary testing set results is similar.

SVM Algorithms :

- During training and testing phases, all of the SVM algorithms require approximately the same amount of memory and CPU power with no significant differences from each other, a trend which was also present in CICIDS 2017 binary dataset results but also in NS-3 binary results and ISCXIDS 2012 dataset results. This demonstrates that regardless of the higher complexity between the different SVM algorithms, their memory usage and CPU power required for training is approximately similar regardless of multi-labelling and/or training size.
- Regarding training time, we observe that all SVM algorithms, need significantly less time than their CICIDS 2017 binary training set equivalent results. Therefore, we conclude that a multi-class training approach benefits the polynomial SVM algorithms (but not SVM(RBF)) in constructing their algorithmic model faster.

- During testing, there are no significant differences with regards to memory usage, CPU power and testing time. Also, when compared to the equivalent results from the CICIDS2017 testing set results, there are no significant differences. This highlights that regardless of the testing size the SVM algorithms are unaffected by it.

KNN Algorithms :

- During both training and testing, all of the KNN algorithms once again need approximately similar memory resources and CPU power regardless of K number and distance measure.
- Furthermore, when compared with the equivalent results from the CICIDS 2017 binary dataset results, we observe that there are no significant differences between the two approaches. Therefore, we conclude that KNN algorithms are not affected with following a multi-class approach.
- With regards to training time, we do not observe any differences between this series of results and the CICIDS binary 2017 training set equivalent results for any of the KNN algorithms.
- With regards to testing time only KNN(Filterd) needs significantly more time than the rest of the ML algorithms. Interestingly, in this series of experiments the testing time needed from KNN(Filterd) was approximately 8 seconds less. This highlights that a multi-class training approach can prove beneficial for KNN algorithms as it reduces the incoming traffic evaluation and classification time.

Tree Algorithms :

- During training and testing phases, random forest requires more memory and CPU power than decision tree algorithm, a trend which was also evident in CICIDS 2017 dataset results. However, their differences cannot be considered highly significant.
- The most notable difference is in training times where random forest needs approximately 4.5 additional seconds than decision tree during training. The same trend was observed in the CICIDS 2017 binary training set equivalent results.

ForChaosLR results and comparison with ML algorithms :

- During training, ForChaosLR requires an additional 0.8-1.8 megabyte and 4-6% CPU power compared to the ML algorithms. However, these differences cannot be considered highly significant.

- This is due to ForChaosLR essentially needing to train two distinct algorithms with different data; ForChaos first-layer needs a small set of legitimate real-time past observational instances while LR second-layer needs a large set of both malicious and legitimate offline instances.
- ForChaosLR needs less training time than more complex ML algorithms such as MLP neural networks, random forest, SVM(Quadratic), SVM(Cubic) and SVM(RBF) and less testing time than more complex ML algorithms such as KNN, SVM(Quadratic), SVM(Cubic) and SVM(RBF) algorithms.
- When this series of results is compared to the CICIDS 2017 dataset results, we observe no major differences and therefore can safely assume that the introduction of a multi-class training approach does not seem to affect ForChaosLR from a hardware requirements point of view.
- However, we have to acknowledge that ForChaosLR requires more memory and CPU power during both training and testing phases than all ML algorithms and more training/testing time than multiple ML algorithms.
- In contrast, ForChaos requires less memory and training/testing time than all the ML algorithms. Once again, ForChaos requires a considerable amount of CPU compared to a number of the ML algorithms during training but less during testing. This is once again due to the reasons already mentioned in Section 9.2.1.

9.3.3 Results Analysis and Discussion for CICIDS 2017 Multi Category Multi-Class Dataset

In Tables 9.11 and 9.12 we present the results for the CICIDS 2017 multi-class multi-category datasets for the training and testing phases for both ForChaosLR multi-layer IDS and the set of ML algorithms.

- For the training phase, none of the ML algorithms require more than approximately 2.6 megabytes and 5.8% of CPU power.
- For the testing phase, none of the ML algorithms require more than approximately 2.2 megabytes and 5.7% of CPU power.
- Regarding training time, we observe SVM(Quadratic), SVM(Cubic), SVM(RBF) and both neural network MLP models taking a considerable longer time than the rest of the ML algorithms.

Table 9.10 CICIDS 2017 Multi-Class Single Category Test Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (seconds)</i>
Bayes Net	1.637	3.341	1.141
Naive Bayes	1.305	3.202	1.022
MLP (0.3)	1.604	5.657	1.132
MLP (0.5)	1.604	3.969	1.100
SVM (Linear)	1.671	4.903	1.108
SVM (Quadratic)	1.687	5.743	1.623
SVM (Cubic)	1.690	5.281	1.872
SVM (RBF)	1.694	5.075	4.464
KNN (K=1, Euclidean)	1.468	5.385	2.396
KNN (K=3, Euclidean)	1.468	4.750	2.583
KNN (K=1, Chebyshev)	1.465	4.386	2.315
KNN (K=3, Chebyshev)	1.465	4.515	2.469
KNN (K=1, Filtered)	1.564	4.913	31.045
KNN (K=3, Filtered)	1.564	5.169	31.318
KNN (K=1, Manhattan)	1.465	4.456	2.540
KNN (K=3, Manhattan)	1.465	4.014	2.730
Decision Tree	1.475	3.198	1.020
Random Forest	1.602	5.057	1.260
ForChaosLR	1.999	7.848	1.166
ForChaos	0.429	3.423	0.089

- When compared to the CICIDS 2017 binary training results we observe that in this series of experiments random forest, SVM(RBF) and both MLP neural networks require significantly more time. This indicates that a multi-class training approach affects all algorithms in a different way with regards to their construction time.
- When compared to the CICIDS 2017 single category multi-class results, the trend followed was similar.
- Regarding testing time, we observe only KNN(Filtered) algorithms once again require the longest time to classify incoming traffic for the reasons already stated and double the time compared to the CICIDS 2017 multi-class single category testing time results.
- The rest of the KNN algorithms however need significantly less time. This trend was also evident in the CICIDS 2017 binary and CICIDS 2017 multi-class single-category results.

Table 9.11 CICIDS 2017 Multi-Class Multi-Category Training Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (seconds)</i>
Bayes Net	2.284	5.724	3.045
Naive Bayes	1.900	3.762	2.271
MLP (0.3)	1.998	5.427	121.022
MLP (0.5)	1.998	5.828	121.595
SVM (Linear)	2.557	3.466	4.783
SVM (Quadratic)	2.557	4.588	16.638
SVM (Cubic)	2.644	4.669	20.915
SVM (RBF)	2.477	5.002	145.295
KNN (K=1, Euclidean)	1.974	4.825	2.334
KNN (K=3, Euclidean)	1.974	4.413	2.258
KNN (K=1, Chebyshev)	1.967	4.988	2.237
KNN (K=3, Chebyshev)	1.967	4.997	2.266
KNN (K=1, Filtered)	2.042	3.532	2.602
KNN (K=3, Filtered)	2.042	3.613	2.563
KNN (K=1, Manhattan)	1.967	4.272	2.383
KNN (K=3, Manhattan)	1.967	4.431	2.248
Decision Tree	2.068	3.691	2.769
Random Forest	2.099	4.508	12.710
ForChaosLR	2.821	8.872	2.546

- Furthermore, in contrast to the CICIDS 2017 single category multi-class results, SVM(RBF) requires a considerable testing time to evaluate the testing set.

Proceeding, we go into more detail regarding the results of the ML algorithms grouped by the family they belong to.

Bayesian Algorithms :

- During both training and testing phases, Bayesian networks algorithm requires more memory, CPU power and training/testing time than naive Bayesian algorithm.
- These trends were also evident in the CICIDS 2017 binary and CICIDS 2017 multi-class single category dataset results. Therefore, we conclude that for all versions of the CICIDS 2017 dataset Bayesian algorithms are unaffected regardless of the training approach (binary v.s. multi-class approach) followed.

Neural Networks Algorithms :

- During both training and testing, both neural network models require approximately the same amount of memory and CPU power, with MLP(0.5) requiring slightly more resources than MLP(0.3).
- The same trends were also observed in the CICIDS 2017 binary and single category multi-class dataset results. This indicates that following a multi-class training approach does not change how neural network MLP algorithms behave when compared with each other.
- Regarding training time, both MLP neural network algorithms need the highest amount of training time compared to the rest of the ML algorithms, except SVM(RBF).
- We have to point out that when compared to the CICIDS 2017 multi-class single category training time equivalent results for both MLP neural network algorithms, the training time is approximately 4 times larger. This indicates that the multi-category multi-class training approach requires the neural network algorithms longer time to construct their algorithmic model now that additional classes were introduced.
- In addition, when compared to the CICIDS 2017 binary training time equivalent results for both MLP neural network algorithms, the training time is approximately 3 times larger. This indicates that the multi-category multi-class training approach requires the neural network algorithms longer time to construct their algorithmic model than a binary approach.

SVM Algorithms :

- During training and testing phases, all of the SVM algorithms require approximately the same amount of memory and CPU power with no significant differences from each other, a trend which was also present in CICIDS 2017 binary and CICIDS 2017 multi-class single category dataset results but also in NS-3 binary results and ISCXIDS 2012 dataset results. This demonstrates that regardless of the higher complexity between the different SVM algorithms and the training approach (binary or multi-class), their memory usage and CPU power required for training is approximately similar.
- Comparing each SVM algorithm's memory usage and CPU power with its CICIDS 2017 multi-class single category dataset equivalent results we observe that all of them require noticeably increased resources. Hence, we observe that the introduction of additional classes to a multi-class training approach evidently requires more memory and CPU resources.

- Furthermore regarding training time, we observe that all SVM algorithms, need significantly more time than their CICIDS 2017 multi-class single category training set equivalent results. Once again, we observe that the introduction of additional classes to a multi-class training approach evidently requires more memory and CPU resources.
- Comparing each SVM algorithm's memory usage and CPU power with its CICIDS 2017 binary dataset equivalent results we observe that all of them require approximately similar resources.
- Furthermore regarding training when comparing with the CICIDS 2017 binary dataset, we observe that, except SVM(RBF), all the SVM algorithms need approximately the same training time while SVM(RBF) needs approximately 2.5 times more training time.
- During testing, there are no significant differences with regards to memory usage, CPU power and testing time. Also, when compared to the equivalent results from the CICIDS 2017 binary and multi-class single category testing set results, there are no significant differences. This highlights that regardless of the testing size the SVM algorithms are unaffected by it. The only difference is in the SVM(RBF) testing time.

KNN Algorithms :

- During both training and testing, all of the KNN algorithms once again need approximately similar memory resources and CPU power regardless of K number and distance measure.
- Furthermore, when compared with the equivalent results from the CICIDS 2017 multi-class single category and binary dataset results, we observe that there are no significant differences between them. Therefore, we conclude that KNN algorithms are not affected with following a multi-class approach.
- With regards to training time, we do not observe any differences between this series of results and the CICIDS 2017 multi-class single category and binary 2017 training set equivalent results for any of the KNN algorithms.
- With regards to testing time, only KNN(Filtered) needs significantly more time than the rest of the ML algorithms. This trend was present in both CICIDS 2017 multi-class single category and binary testing set results.

- In this series of results, we observe that KNN(Filtered) needs significantly more time than the equivalent results from the CICIDS 2017 multi-class single category testing set (approximately 30 additional seconds) and the CICIDS 2017 binary testing set (approximately 20 seconds). This denotes that the introduction of additional classes impacted the classification time for KNN(Filtered) algorithm.

Tree Algorithms :

- During training and testing phases, random forest requires more memory and CPU power than decision tree algorithm, a trend which was also evident in CICIDS 2017 binary and single category multi-class dataset results. However, their differences cannot be considered highly significant.
- The most notable difference is in training time where random forest needs approximately 10 additional seconds than decision tree during training.
- Compared with the equivalent results from the CICIDS 2017 multi-class single category and binary training set results we observe that the training time for random forest was increased. This is due to the introduction of additional classes which clearly an impact in training time as the trees become more complex.

ForChaosLR results and comparison with ML algorithms :

- During training, ForChaosLR requires an additional 0.7-0.9 megabyte and 3.1-5.4% CPU power compared to the ML algorithms. However, these differences cannot be considered highly significant.
- This is due to ForChaosLR essentially needing to train two distinct algorithms with different data; ForChaos first-layer needs a small set of legitimate real-time past observational instances while LR second-layer needs a large set of both malicious and legitimate offline instances.
- However, the training and testing times for ForChaosLR were not higher than the highest training and testing times of the ML algorithms. Therefore, ForChaosLR needs less training time than more complex ML algorithms such as Bayesian networks, MLP neural networks, decision tree, KNN(Filtered), random forest, SVM(Quadratic), SVM(Cubic) and SVM(RBF) and less evaluation time than more complex ML algorithms such as KNN, SVM(Quadratic), SVM(Cubic) and SVM(RBF) algorithms.

- Concluding we observe that ForChaosLR requires more memory and CPU power than all ML algorithms during both training/testing phases and more training/testing time than multiple ML algorithms.
- In contrast, ForChaos requires less memory and training/testing time than all the ML algorithms. Once again, ForChaos requires a considerable amount of CPU compared to a number of the ML algorithms during training but less during testing. This is once again due to the reasons already mentioned in Section 9.2.1.

Table 9.12 CICIDS 2017 Multi-Class Multi-Category Test Set Hardware Requirements Metrics

	<i>Memory Usage (Megabytes)</i>	<i>CPU Power (%)</i>	<i>Time Elapsed (seconds)</i>
Bayes Net	1.637	4.942	1.165
Naive Bayes	1.335	4.289	1.088
MLP (0.3)	1.604	4.700	1.122
MLP (0.5)	1.604	4.764	1.132
SVM (Linear)	1.671	3.457	1.133
SVM (Qadratic)	1.696	5.201	2.594
SVM (Cubic)	2.192	4.892	3.235
SVM (RBF)	2.116	5.672	12.876
KNN (K=1, Euclidean)	1.468	5.052	3.732
KNN (K=3, Euclidean)	1.468	4.278	3.971
KNN (K=1, Chebyshev)	1.465	4.770	3.703
KNN (K=3, Chebyshev)	1.465	5.083	3.865
KNN (K=1, Filtered)	1.587	5.001	60.015
KNN (K=3, Filtered)	1.587	4.894	60.116
KNN (K=1, Manhattan)	1.465	4.023	4.013
KNN (K=3, Manhattan)	1.465	5.061	4.232
Decision Tree	1.475	5.035	0.976
Random Forest	1.742	5.490	1.264
ForChaosLR	1.999	8.942	1.182
ForChaos	0.429	3.423	0.089

9.4 Summary

In this chapter, we presented all of the Raspberry Pi 3 model B+ evaluation experimental results for ForChaosLR multi-layer IDS and the ML algorithms chosen. Different metrics

were measured; memory usage, CPU power and time to execute. All the datasets except the smart home datasets were used to track these metrics for ForChaosLR as well as the ML algorithms chosen. In general, we observe that none of the ML algorithms or ForChaosLR overstress the Raspberry Pi 3 model B+ device up to the point it output a warning and/or shuts down.

In general, we observe that all of the ML algorithms and ForChaosLR need more memory, CPU power and time during training than in testing. This was expected as constructing an algorithmic model is lengthier in time and demands more resources. The only exception to this finding was the set of KNN algorithms however, this was also expected as KNN does not consist of an algorithmic construction during "training phase" and rather performs classification "on the fly".

Regarding binary training, we observe that the absence of slow-rate instances in ISCXIDS 2012 dataset makes the ML algorithms and ForChaosLR less demanding in training time, memory usage and CPU power during training phase. Hence, we observe that the construction of an accurate algorithmic model that can differentiate between DDoS (flooding and slow-rate) and legitimate traffic is a lengthier and more complex process. This is also reflected during the multi-class training phases where the training time, memory usage and CPU power requirements are also increased. Furthermore, we also observe the training set's size play a significant role in training phase for all ML algorithms and ForChaosLR regardless of the training being binary or multi-class.

During training and testing, ForChaosLR requires more memory and CPU power regardless of dataset used. In addition, ForChaosLR require more time during training/testing than the majority of the ML algorithms, with some exceptions that mainly involve the SVM algorithms and neural networks during training and SVM and KNN algorithms during testing. For more details regarding these exceptions please refer to the specific experiment with the specific dataset in this Chapter.

On the other hand, ForChaos first-layer requires less memory usage, execution time and CPU power during testing and less memory usage and execution time but more CPU power during training than ForChaosLR.

With regards to training, ForChaos first-layer only requires a very small amount of training size which only consists of legitimate instances to construct its prediction model. This comes in contrast to all the other ML algorithms used, including LR second-layer of ForChaosLR, which require a considerable size of a training set consisting of both legitimate and attack instances to construct their models. Hence why, ForChaos first-layer requires less memory and execution time. As already stated, ForChaos needs more CPU power during training than in testing since it needs to evaluate multiple combinations α and window sizes

to find the most optimal one using the RMSE value. More information regarding the training procedure of ForChaos first-layer can be found in Chapter 4, Section 1.

Therefore, based on the acquired results presented throughout this Chapter, it is LR second-layer which requires a considerable amount of memory, CPU power and execution time during both training and testing phases. This is evident for both binary and multi class classification. For the binary classification, LR second-layer requires more resources due to the exponential calculations required based on its equation needed. For the multi-class class classification, logistic regression constructs a number of logistic regression models which is equal to the total number of labels in the training set provided. Therefore, during testing each incoming instance is going to be processed through every logistic regression model and the one that achieves the highest probability assigns its label to the instance.

The main reason for logistic regression chosen as the algorithm for the second-layer of ForChaosLR was due to the literature suggesting that it is suitable for substantially decreasing FP rates as described in Chapter 3, Section 3.5. However, it is evident, both from the results and the overall functioning of the algorithm itself, that logistic regression would not be the most suitable solution for a resource-constrained device such as IoT devices.

Chapter 10

Conclusion

10.1 Overview

In the final chapter of the thesis, an overview of the complete study is presented by summarising our contributions. The main contributions provided from this thesis are presented. Proceeding, we provide a list of future work and improvements the thesis can approach. In the final section of the chapter, we outline concluding remarks and summarise our main work and what lessons were learnt throughout the thesis.

10.2 Summary of Contributions

The main contributions provided from our thesis include the following:

1. *Comprehensive and in-depth research on the state of the art regarding intrusion detection in IoT and resource-constrained networks.*

An in-depth investigation of analysis was conducted in intrusion detection. In detail, the investigation targeted intrusion detection solutions that focused on multiple topics that we were interested in our research. These topics include DDoS detection in all TCP/IP layers, application layer DDoS detection, intrusion detection using lightweight techniques in resource constrained networks such as IoT, intrusion detection using forecasting techniques, chaos theory, regression and other machine learning and A.I. techniques and intrusion detection following a multi-layer approach. Finally, after summarising the key finding related to these studies, we identified the literature gaps and limitations when applied to our research study.

2. Simulation of the smart home-smart grid network and construction of a smart home-smart grid dataset

Based on the knowledge regarding the smart home-smart grid network acquired, we developed a simulation of the smart home-smart grid network based on the relevant and latest standards, specifications and protocols used on NS-3 network simulator. Specifically, we implemented additional libraries in NS-3 for IoT application layer protocols such as CoAP, MQTT and XMPP as well as HTTP. Our HTTP implementation was inspired by [195]. Proceeding, we simulated flooding and slow-rate DDoS attacks in the smart home-smart grid network as well as flash crowd events by implementing relevant code in libraries. These relevant implementations are available upon request. Finally, we generated and collected enough data to construct a dataset to use for the ForChaosLR multi-layer IDS's training and evaluation. The dataset is available for relevant research and experimental evaluation upon request.

3. Design, implementation and evaluation of ForChaosLR multi-layer IDS for the smart home-smart grid network to detect application layer flooding and slow-rate DDoS attacks

We proposed a lightweight multi-layer approach for the smart home-smart grid aiming to detect application layer flooding and slow-rate DDoS attacks. Firstly, we presented ForChaosLR multi-layer IDS using the relevant mathematical concepts and proofs. Next, we described how and under what circumstances ForChaosLR multi-layer IDS should operate and function. Finally, we outlined the set of experiments ForChaosLR multi-layer IDS solution was to be evaluated.

ForChaosLR multi-layer IDS is a solution specifically for the smart home-smart grid network. Using the dataset generated from the smart home-smart grid network simulation in NS-3, we simulated a series of experiments to test our proposed solution. Through our experiments we introduced different scenarios, focusing on application layer DDoS flooding and slow-rate attacks described more in detail in Chapter 5. Furthermore, to gain more in depth results we analysed and used other real-world datasets that consist of application layer flooding and slow-rate DDoS attacks, also described more in detail in Chapter 5. Finally, we deployed ForChaosLR multi-layer IDS and other machine learning algorithms so we could evaluate hardware performance and assess the suitability of the Raspberry Pi 3.

4. Design, Implementation and Evaluation of ForChaosLR multi-layer IDS for the smart home-smart grid network to perform multi-attack classification

By extending our work in the previous contribution, we evaluated ForChaosLR multi-layer IDS to not only detect malicious behaviour (application layer flooding and slow-rate DDos attacks) but also classify incoming detected malicious behaviour as either flooding or slow-rate. Furthermore, when multi-attack classification was performed using the real-world datasets more than the two attacks considered, specifically Slowloris, SlowHTTPTest, HULK and Goldeneye [92]. Based on the results gained, we observe that ForChaosLR multi-layer IDS can detect these types of slow-rate attacks with the set of features used only for the flooding and slow-rate (slowloris) attacks. Finally, we deployed ForChaosLR multi-layer IDS and other machine learning algorithms so we could evaluate hardware performance and assess the suitability of the Raspberry Pi 3 model B+.

10.3 Limitations and Future Work

The main objective of the thesis was for ForChaosLR to be accurate in detecting flooding and slow-rate DDoS attacks in IoT networks such as the smart home-smart grid but also lightweight compared to popular ML algorithms. ForChaosLR produced very good results with IoT network traffic datasets with regards to accuracy.

Specifically, it achieved a 97.300% TP rate and a 0.52% FP rate when the NS-3 binary dataset was used and a 97.744% TP rate and a 0.056% FP rate when the NS-3 multi-class dataset was used. In addition, ForChaosLR achieved only 0.4% and 0.2% FP rates for the 2018 and 2019 smart homes datasets. More information regarding the datasets can be found in Chapter 5, Sections 5.2.1, 5.2.3. Comparing the NS-3 results (both binary and multi-class), we observe that ForChaosLR produces a good TP rate, although it was not the highest but still among the highest rates. With regards to the FP rates, in the former dataset ForChaosLR produces one of the lowest FP rates and in the latter, the lowest. In addition, when ForChaosLR is evaluated with the smart home datasets it maintains a very low FP rate.

ForChaosLR produced satisfactory results on datasets from other independent universities such as the one presented in Chapter 5, Section 5.2.2, 5.2.2, 5.2.3, 5.2.3. This indicates that ForChaosLR is able to classify different variations of flooding and slow-rate DDoS attacks such as Goldeneye, HULK and SlowHTTPTest as well as identify legitimate high variations of network traffic. However, the majority of the ML algorithms produced better results, specifically higher TP rate and/or lower FP rate.

In addition, although we aimed for a more lightweight approach compared to the ML algorithms we observe that ForChaosLR requires more memory usage and CPU power during both training/testing phases when deployed on a Raspberry Pi 3 model B+ compared

to the ML algorithms. On the other hand, as demonstrated throughout the experimental evaluation in Chapter 9, FoChaosLR requires less training time than MLP neural networks and SVM algorithms (except Linear) and less testing time than KNN and SVM (except Linear) algorithms mainly. More details on the experimental evaluation results are presented and explained in Chapter 9.

However, we have to point out that ForChaos first-layer requires less memory usage, execution time and CPU power during testing and less memory usage and execution time but more CPU power during training than ForChaosLR as demonstrated throughout Chapter 9.

In more detail, ForChaos first-layer only requires a very small amount of training size which only consists of legitimate instances to construct its prediction model. This comes in contrast to all the other ML algorithms used, including LR second-layer of ForChaosLR, which require a considerable size of a training set consisting of both legitimate and attack instances to construct their models. Hence why, ForChaos first-layer requires less memory and execution time. As already stated, ForChaos needs more CPU power during training than in testing since it needs to evaluate multiple combinations α and window sizes to find the most optimal one using the RMSE value. More information regarding the training procedure of ForChaos first-layer can be found in Chapter 4, Section 1.

Therefore, it is LR second-layer which requires a considerable amount of memory, CPU power and execution time during both training and testing phases. This is evident for both binary and multi class classification demonstrated in Chapter 9. For the binary classification, LR second-layer requires more resources due to the exponential calculations required based on its equation needed. For the multi-class class classification, logistic regression constructs a number of logistic regression models which is equal to the total number of labels in the training set provided. Therefore, during testing each incoming instance is going to be processed through every logistic regression model and the one that achieves the highest probability assigns its label to the instance.

The main reason for logistic regression chosen as the algorithm for the second-layer of ForChaosLR was due to the literature suggesting that it is suitable for substantially decreasing FP rates as described in Chapter 3, Section 3.5. However, it is evident, both from the results and the overall functioning of the algorithm itself, that logistic regression would not be the most suitable solution for a resource-constrained device such as IoT devices.

Although ForChaosLR is not the most lightweight algorithm when compared to the ML algorithms implemented, we conclude that it demonstrates a series of advantages compared to the ML algorithms which we proceed by briefly describing them.

- ForChaos first-layer only requires a very small amount of instances to construct its algorithmic model unlike ML algorithms. Specifically, ForChaos first-layer only

needs 25 legitimate traffic instances to construct an algorithmic model with very good accuracy as demonstrated throughout Chapters 6, 7 and 8. In contrast, the ML algorithms need a considerable size of training set to construct their models. Specifically, for the NS-3 binary training set the ML algorithms needed 3757 attack and 3230 legitimate instances. For the ISCXIDS 2012 training set, the ML algorithms required 2548 attack and 3712 legitimate instances. For the CICIDS 2017 binary training set the ML algorithms required 1862 attack and 1119 legitimate instances. For the multi-class NS-3 training set, the ML algorithms required 1896 flooding, 1868 slow-rate and 3228 legitimate instances. For the CICIDS 2017 single category multi-class training set, the ML algorithms required 840 flooding, 1055 slow-rate and 440 legitimate instances. Finally, for the CICIDS 2017 multi category multi-class training set, the ML algorithms required 840 flooding, 924 slow-rate, 464 slowhttptest, 720 hulk, 806 goldeneye and 880 legitimate instances. We have to acknowledge that LR second-layer was also trained with the same training sets as the ML algorithms. However, even without the second layer, ForChaos first-layer can be developed and deployed to detect DDoS attacks on its own. On the other hand, the ML algorithms do not have this capability. Therefore, ForChaos first-layer can be deployed faster than the ML algorithms, with less training size and has the capability to be re-trained more frequently.

- Extending our previous point, ForChaos first-layer only needs legitimate traffic to construct its model, an important advantage compared to the ML algorithms used. In the real-world, providing a considerable size of malicious traffic might not always be possible. A notable example in cyber security where this is the case is the detection of credit card/financial transactions fraud behaviour [254], [45], [28], [48], [200], [111], [182], [212].

The authors in [254], [28] effectively highlighted, the available datasets (whether they are public or private) are heavily imbalanced due to lack of fraud detection data. Imbalanced data as described in Chapter 6, Section 6.7, can cause multiple issues during training, resulting into the algorithm not constructing an accurate model. ForChaos first-layer bypasses that issue since it does not require malicious behaviour to construct its model.

- Finally, explainable artificial intelligence has been gaining a lot of attention recently. Its application on cyber security can be beneficial as the authors in [106], [143], [23] effectively supported. Furthermore, they argued that not enough explanation is provided with regards to an IDS's results due to the ML algorithms' black box

nature. The black box nature would not likely be an issue for ForChaosLR. The first layer which would consist of forecasting algorithms and Lyapunov exponents. Proper explanations on the error between the predictions of the incoming traffic and their actual numbers can be provided to gain more insights on what triggered the Lyapunov exponent to be positive. In addition, logistic regression operates with probabilities. The actual probability along with its relevant calculations could be provided to the human personnel in charge to improve mitigation actions.

1. *Improvement of ForChaosLR resources requirements*

One of the most important objectives of ForChaosLR that was not fully met was ForChaosLR being a lightweight alternative to popular ML algorithms. Unfortunately, this objective was not possible with the current implementation of ForChaosLR. In detail, ForChaosLR required more memory and CPU power than all ML algorithms during both training and testing phases for all the datasets it was evaluated with. With regards to training/testing time, ForChaosLR required less training time than SVM (except Linear) and MLP neural networks and less testing time than KNN and SVM (except Linear) algorithms. However, this was not enough to qualify ForChaosLR as a lightweight alternative to the other ML algorithms. The problem as explained in Chapter 9, Section 9.4 lied on the logistic regression algorithm (LR second-layer) and not on ForChaos. In the future, new algorithms can be evaluated for the second layer which can effectively reduce the FP rates but also not cause such a significant overhead to the resource-constrained device.

2. *DDoS attacks detection in multiple layers*

Our research aimed to detect application layer DDoS attacks as the relevant literature was not rich in studies compared to DDoS attacks detection in lower layers. An improvement for the system would be to secure it against DDoS attacks in lower layers as well. An adversary could combine multiple DDoS types of attacks from multiple layers for two main reasons: to ensure the negative impact on target and also remain undetected due to malicious behaviour being initiated from multiple layers. In that way, a sophisticated IDS to secure against DDoS attacks regardless of the layer should be developed.

3. *Other network attacks detection*

As already stated in Improvement 1, the expansion of DDoS detection in multiple layers is a very important step. To expand on the previous Improvement in further, we could extend ForChaosLR multi-layer IDS to detect other types of malicious behaviour. Some

examples include botnet detection during the infection and propagation phase, illegal file download, IP/MAC spoofing, MITM attacks as well as malicious surveillance behaviour. Furthermore, cyber-physical attacks could be explored such as energy stealing attacks. In order for Improvements 1 and 2 to be successfully met, additional features should be added to the ForChaosLR multi-layer IDS. New additional features can be introduced to the system for achieving accurate detection and minimise the FP rates. However, as multiple studies in the field of intrusion detection supported by [243], [256], [231], [110], [184], [20], [177] and [221] using a large number of features is going to increase the system complexity may have a negative impact towards correct classification. Therefore, a feature reduction without compromising the IDS's accuracy can be a very important step towards its optimisation. Furthermore, the optimisation of features can ensure that the complexity is not increased, something that is highly important on the smart home-smart grid network.

4. *Expansion of the NS-3 dataset*

Another important step for the success of the two previous suggested Improvements, is the expansion of the dataset. Since, more types of attacks will be considered for detection, they have to be firstly simulated and then introduced to the ForChaosLR multi-layer IDS for training the LR second-layer specifically. Then, ForChaosLR multi-layer IDS must be evaluated against these kind of attacks to test its accuracy. Therefore, the dataset should be expanded so it can include these types of attacks as well. Furthermore, certain extensions based on the scenarios already constructed can be constructed. Examples include the construction of flash crowd scenarios followed by DDoS attack or restart DDoS scenarios in which the first attack is not the same as the second attack (e.g. flooding then slow-rate and vice versa).

5. *Intrusion detection in IoT smart cities*

Our research focused on the smart home-smart grid network as their interaction is one of the most important ones in the smart city context. However, there are other interactions between other IoT networks in the smart city which are just as important. Therefore, a possible future direction would include to adapt the ForChaosLR multi-layer IDS to other IoT networks in the smart city. Some examples could include VANETs (vehicular area networks), intelligent transportation systems, smart airports, smart watering systems and so on.

6. *Secure and reliable communication of the two IDS systems through authentication and encryption*

In IDS systems where multiple detection agents are involved a compromise of one or more of the agents can be done. Therefore, we need to ensure reliable and secure communication. Possible future work would include to implement trust management and authentication mechanisms. These mechanisms can assist in identifying the systems and ensure legitimate communication between them.

7. *Defend against adversarial machine learning*

8. Adversarial machine learning is another important topic that has gained a lot of recognition recently in cyber security [106], [143], [23]. We have to acknowledge that an adversary can follow all the relevant adversarial machine learning techniques to bypass ForChaosLR and conduct the DDoS attack without being detected. Future work would include modelling such scenarios and proceeding with defending ForChaosLR against such malicious behaviour.

10.4 Conclusions and lessons learnt

A critical, if not the most important, technology integrated in smart cities is the IoT. It provides a wide range of opportunities and potential to every cyber-physical system deployed. IoT is particularly beneficial to the interactions between the smart homes (or smart buildings in general) and the smart grid. However, this tight interconnection between them can cause security issues and enable the conduction of cross-network attacks. As availability is vital in critical infrastructures, DDoS attacks is an ideal attack to threaten this cyber security principle. DDoS attacks in the application layer became popular recently due to their effectiveness and lower in resources requirements when compared to DDoS attacks in lower layers. IoT devices are unfortunately known for their weak or complete lack of security countermeasures. Hence, it is easier for an adversary to compromise them and construct a botnet to perform large in magnitude DDoS attacks.

As a result, the deployment of intrusion detection systems on the edge of IoT networks is very important. However, an IDS should be lightweight, in addition to accurate. This motivated us to propose ForChaosLR multi-layer IDS, a multi-layer IDS designed specifically for resource-constrained devices in IoT networks.

However, our proposed IDS was not the most lightweight when compared to the ML algorithms chosen as observed in the experimental results throughout Chapter 9.

Nevertheless, our chosen approach is able to detect both flooding and slow-rate application Layer DDoS attacks as demonstrated throughout Chapters 6, 7 and 8 experimental results. We have to denote that ForChaosLR did not achieved the highest accuracy but it consisted

one of the highest accuracy rates, with the exception of the CICIDS 2017 multi category multi-class dataset results.

Based on our experimental evaluations, we benefited with multiple conclusions. In summary:

- Firstly, we demonstrated our ability to propose a new method for detection of application layer flooding and slow-rate DDoS attacks, implement it and evaluate it by comparing it with popular ML algorithms.
- The combination of simple exponential smoothing forecasting and Lyapunov exponents (ForChaos first-layer) was able to detect flooding and slow-rate DDoS attacks in the smart home-smart grid IoT network, as demonstrated in Chapters 6. Specifically, its TP rate was higher than MLP neural networks, SVM(RBF) and all KNN algorithms (except Filtered K=1). Its FP rate was only higher than SVM(Cubic) and SVM(RBF). As already stated, although it did not achieved the highest accuracy, its results are positive and reassuring. This was also the case in the multi-class NS-3 dataset results, where ForChaosLR achieved one of the highest TP rates (only lower than Bayesian network, naive Bayesian and decision tree algorithms) and the lowest FP rate.
- ForChaosLR was also effective in detecting variations of slow-rate and flooding attacks as effectively demonstrated in Chapter 7. Its TP rate was only lower from Bayesian network and KNN (Euclidean and Manhattan) algorithms. However, we have to point out that its FP rate was also considerably high (4.53%) only lower than naive Bayesian and all SVM algorithms.
- On the other hand, ForChaosLR was not improved when a multi-class training approach was followed for the CICIDS 2017 dataset as demonstrated in Chapter 7 experimental results. Although its TP rate remained unaffected when compared to the equivalent CICIDS 2017 binary dataset rate, the ML algorithms' respective TP rates were improved, thus achieving a higher TP rate than ForChaosLR. Furthermore, the ML algorithms were able to correctly identify the specific attack type with a higher accuracy than ForChaosLR. This indicates that logistic regression demonstrates issues with handling multi-class problems with high accuracy as the other ML algorithms.
- Based on literature, logistic regression is shown to produce substantially low FP rates. This finding was also confirmed in our results in Chapter 6, 7 and 8 as LR second-layer manages to drastically reduce the FP rates achieved from ForChaos first-layer whether or not a re-training phase was introduced for the latter. Therefore, our results contribute

to the already existing literature suggesting that logistic regression is a suitable solution for reducing FP rates as demonstrated in Chapter 3, 3.5.

- ForChaosLR multi-layer IDS demonstrated to effectively classify flash crowd traffic as legitimate. This finding indicates that the combination of ForChaosLR consists of a ForChaos first-layer (effective against anomalous traffic algorithm) and LR second-layer (effective in classifying normal spikes as legitimate) works beneficially in classifying flash crowd events is successful. Our results contribute to the already existing literature, researching on how to differentiate between flooding and flash crowd behaviour such as [251], [217], [248] and [81].
- Finally, we have to acknowledge that although the combination of forecasting/Lyapunov exponents and logistic regression, although it produced satisfactory results from an accuracy point of view, did not proved to be a lightweight alternative. However, as already stated, we believe that with some notable changes to its implementation, we will be able to substantially reduce its hardware requirements.
- Although ForChaosLR consists of essentially two algorithms that have different training approaches, when deployed on a Raspberry Pi 3 model B+, it does demonstrate significant hardware resources differences when compared to the rest of the ML algorithms. This indicates that the combination of two layers that however, do not work simultaneously is beneficial to Raspberry Pi 3 model B+ as it does not overstress it. However, ForChaos first-layer on its own is indeed a lightweight algorithm and the overhead issues lies on the LR second-layer for the reasons explained in Chapter 9, Summary Section 9.4. Hence, although the literature correctly suggests that a logistic regression model can reduce FP rates (which was also confirmed through our own experiments in Chapters 6 and 7) it can also be rather high in resources.

References

- [1] A. Fernández, S. Garcia, F. v. and Chawla, N. (2018). Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61:863–905.
- [2] A. Furtuna (2010). Ddosim - layer 7 ddos simulator. SourceForge. [online]Available: [adrianfurtuna](#).
- [3] A. Hafeez, N. H. Kandil, A. B. T. L. and Al-Ali, A. (2014). Smart home area networks protocols within the smart grid context. *Journal of Communications*, 9(9):665–671.
- [4] A. Hamza, H. H. Gharakheili, T. A. B. and Sivaraman, V. (2019). Detecting volumetric attacks on iot devices via sdn-based monitoring of mud activity. In *Proceedings of the 2019 ACM Symposium on SDN Research, SOSR '19*, page 36–48, New York, NY, USA. Association for Computing Machinery.
- [5] A. Shiravi, H. Shiravi, M. T. and Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357–374.
- [6] Ahemd, M. M., Shah, M. A., and Wahid, A. (2017). Iot security: A layered approach for attacks defenses. In *2017 International Conference on Communication Technologies (ComTech)*, pages 104–110.
- [7] Ajagekar, S. K. and Jadhav, V. (2016). Study on web ddos attacks detection using multinomial classifier. In *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pages 1–5.
- [8] Ali, M. Q. and Al-Shaer, E. (2013). Probabilistic model checking for ami intrusion detection. In *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 468–473.
- [9] Allen, D. M. (1974). The relationship between variable selection and data augmentation and a method for prediction. *Technometrics*, 16(1):125–127.
- [10] Almeida, V. A. F., Doneda, D., and de Souza Abreu, J. (2017). Cyberwarfare and digital governance. *IEEE Internet Computing*, 21(2):68–71.
- [11] Almon, L., Riecker, M., and Hollick, M. (2017). Lightweight detection of denial-of-service attacks on wireless sensor networks revisited. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 444–452.

- [12] Aloul, F., A.R., Al-Ali, Al-Dalky, R., Al-Mardini, M., and El-Hajj, W. (2012). Smart grid security: Threats, vulnerabilities and solutions. *International Journal of Smart Grid and Clean Energy*, 1(1):1–6.
- [13] Alseiyari, F. A. A. and Aung, Z. (2015). Real-time anomaly-based distributed intrusion detection systems for advanced metering infrastructure utilizing stream data mining. In *2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*, pages 148–153.
- [14] Altman, D. G. and Bland, J. (1994). Diagnostic tests. 1: Sensitivity and specificity. *BMJ: British Medical Journal*, 308(6943):1552.
- [15] A.Mayzaud, R. B. and Chrisment, I. (2016). A taxonomy of attacks in rpl-based internet of things. *International Journal of Network Security*, 18:459–473.
- [16] Amin, P., Cherkasova, L., Aitken, R., and Kache, V. (2019). Automating energy demand modeling and forecasting using smart meter data. In *2019 IEEE International Congress on Internet of Things (ICIOT)*, pages 133–137.
- [17] Andrea, I., Chrysostomou, C., and Hadjichristofi, G. (2015). Internet of things: Security vulnerabilities and challenges. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 180–187.
- [18] Anthi, E., Williams, L., and Burnap, P. (2018). Pulse: An adaptive intrusion detection for the internet of things. In *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–4.
- [19] Aravind, M. A. M. and Kalaiselvi, V. K. G. (2017). Design of an intrusion detection system based on distance feature using ensemble classifier. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–6.
- [20] Aziz, A. S. A., Hassanien, A. E., Hanaf, S. E., and Tolba, M. F. (2013). Multi-layer hybrid machine learning techniques for anomalies detection and classification approach. In *13th International Conference on Hybrid Intelligent Systems (HIS 2013)*, pages 215–220.
- [21] B. Shteiman (2012). Hulk, http unbearable load king. Packet Storm. [online]Available: <https://goo.gl/PWhEJk>.
- [22] Banks, A. and Gupta, R. (2014). Mqtt version 3.1. 1. *OASIS standard*, 29.
- [23] Baracaldo, N., Chen, B., Ludwig, H., Safavi, A., and Zhang, R. (2018). Detecting poisoning attacks on machine learning in iot environments. In *2018 IEEE International Congress on Internet of Things (ICIOT)*, pages 57–64.
- [24] Basu, S. S., Tripathy, S., and Chowdhury, A. R. (2015). Design challenges and security issues in the internet of things. In *2015 IEEE Region 10 Symposium*, pages 90–93.
- [25] Behal, S. and Saluja, K. (2017). Characterization and comparison of ddos attack tools and traffic generators -a review. *International Journal of Network Security*, 19:383–393.
- [26] Beigi-Mohammadi, N., Mišić, J., Khazaei, H., and Mišić, V. B. (2014). An intrusion detection system for smart grid neighborhood area network. In *2014 IEEE International Conference on Communications (ICC)*, pages 4125–4130.

- [27] Beitollahi, H. and Deconinck, G. (2013). Connectionscore: a statistical technique to resist application-layer ddos attacks. *Journal of Ambient Intelligence and Humanized Computing*, 5(3):425–442.
- [28] Benchaji, I., Douzi, S., and ElOuahidi, B. (2018). Using genetic algorithm to improve classification of imbalanced datasets for credit card fraud detection. In *2018 2nd Cyber Security in Networking Conference (CSNet)*, pages 1–5.
- [29] Bhunia, S., Hsiao, M. S., Banga, M., and Narasimhan, S. (2014). Hardware trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247.
- [30] Bhunia, S. S. and Gurusamy, M. (2017). Dynamic attack detection and mitigation in iot using sdn. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6.
- [31] Bhushan, K. and Gupta, B. B. (2018). A novel approach to defend multimedia flash crowd in cloud environment. *Multimedia Tools and Applications*, 77:1–31.
- [32] Bishop, C. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*.
- [33] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.
- [34] Bravo, S. and Mauricio, D. (2018). Ddos attack detection mechanism in the application layer using user features. In *2018 International Conference on Information and Computer Technologies (ICICT)*, pages 97–100.
- [35] Brown, R. (1956). *Exponential Smoothing for Predicting Demand*. Little Publishers.
- [36] Burnham, K. P. and Anderson, D. R. (2003). Model selection and multimodel inference : a practical information-theoretic approach. *Journal of Wildlife Management*, 67:655.
- [37] Butun, I., Morgera, S. D., and Sankar, R. (2014). A survey of intrusion detection systems in wireless sensor networks. *IEEE Communications Surveys and Tutorials*, 16(1):266–282.
- [38] C. Bormann, M. E. and Keranen, A. (2014). Terminology for constrained-node networks. *RFC*, 7228:1–17.
- [39] C. F. Gauss, C. H. D. and of America Project, M. (1857). *Theory of the motion of the heavenly bodies moving about the sun in conic sections a translation of Gauss's "Theoria motus." With an appendix*. Boston, Little, Brown and company,.
- [40] Cam-Winget, N., Sadeghi, A.-R., and Jin, Y. (2016). Invited - can iot be secured: Emerging challenges in connecting the unconnected. In *Proceedings of the 53rd Annual Design Automation Conference, DAC '16*, New York, NY, USA. Association for Computing Machinery.
- [41] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and Dijk, E. (2017). Guidelines for mapping implementations: http to the constrained application protocol (coap). *Internet Engineering Task Force (IETF): Fremont, CA, USA*, pages 2070–1721.

- [42] Cawley, G. C. and Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.*, 11:2079–2107.
- [43] Chandrasekaran, B. (2006). Survey of network traffic models.
- [44] Chen, C. H., Lin, I. C., and Yang, C. C. (2014). Nfc attacks analysis and survey. In *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 458–462.
- [45] Chen, Y. and Han, X. (2021). Catboost for fraud detection in financial transactions. In *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pages 176–179.
- [46] Chen, Y., Ma, X., and Wu, X. (2013). Ddos detection algorithm based on preprocessing network traffic predicted method and chaos theory. *IEEE Communications Letters*, 17(5):1052–1054.
- [47] Chen, Y., Ma, X., and Wu, X. (2013). Ddos detection algorithm based on preprocessing network traffic predicted method and chaos theory. *IEEE Communications Letters*, 17(5):1052–1054.
- [48] Chen, Y.-J. and Wu, C.-H. (2017). On big data-based fraud detection method for financial statements of business groups. In *2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 986–987.
- [49] Chen Xiuqing, Zhang Yongping, and Tang Jiutao (2010). Hmm-based integration of multiple models for intrusion detection. In *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 2, pages V2–137–V2–140.
- [50] Chonka, A., Singh, J., and Zhou, W. (2009). Chaos theory based detection against network mimicking ddos attacks. *IEEE Communications Letters*, 13(9):717–719.
- [51] Chonka, A., Singh, J., and Zhou, W. (2009). Chaos theory based detection against network mimicking ddos attacks. *IEEE Communications Letters*, 13(9):717–719.
- [52] CISCO (2019). Redefine connectivity by building a network to support the internet of things. Technical report.
- [53] Cleveland, F. M. (2008). Cyber security issues for advanced metering infrastructure (ami). In *2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, pages 1–5.
- [54] Coppolino, L., DAntonio, S., Garofalo, A., and Romano, L. (2013). Applying data mining techniques to intrusion detection in wireless sensor networks. In *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 247–254.
- [55] Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242.

- [56] D. C. Montgomery, E. A. P. and Vining, G. G. (2012). *Introduction to linear regression analysis*, volume 821. John Wiley & Sons.
- [57] D. Casciani (2019). Briton who knocked liberia offline with cyber attack jailed. BBC News. [online]Available: <https://bbc.co.uk/news/uk-46840461>.
- [58] D. Gusak, A. Kukush, A. K. Y. M. A. P. (2010). *Theory of Stochastic Processes With Applications to Financial Mathematics and Risk Theory*. Springer Science and Business Media.
- [59] Darby, S. (2006). The effectiveness of feedback on energy consumption. *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, 486.
- [60] Deakin, M. and Waer, H. A. (2011). From intelligent to smart cities. *Intelligent Buildings International*, 3(3):140–152.
- [61] Deng, R., Yang, Z., Chow, M.-Y., and Chen, J. (2015). A survey on demand response in smart grids: Mathematical models and approaches. *IEEE Transactions on Industrial Informatics*, 11(3):570–582.
- [62] Deogirikar, J. and Vidhate, A. (2017). Security attacks in iot: A survey. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 32–37.
- [63] Devaney, R. L. (2003). *An Introduction to Chaotic Dynamical Systems (2nd ed.)*. Westview Press.
- [64] Dilraj, M., Nimmy, K., and Sankaran, S. (2019). Towards behavioral profiling based anomaly detection for smart homes. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pages 1258–1263.
- [65] Economic and Council, S. R. (2018). Smart cities and sustainability. Technical report.
- [66] Elaydi, S. N. (2007). *Discrete chaos: with applications in science and engineering*. Chapman and Hall/CRC.
- [67] Elisan, C. and Hypponen, M. (2013). *Malware, rootkits & botnets: A beginner's guide*. McGraw-Hill.
- [68] Esmaily, J., Moradinezhad, R., and Ghasemi, J. (2015). Intrusion detection system based on multi-layer perceptron neural networks and decision tree. In *2015 7th Conference on Information and Knowledge Technology (IKT)*, pages 1–5.
- [69] et al., S. G. (2021). Signaling traffic in internet-of-things mobile networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 452–458.
- [70] F. Kandah, Y. Singh, W. Z. and Wang, C. (2013). Mitigating colluding injected attack using monitoring verification in mobile ad-hoc networks. *Security and Communication Networks*, 6(4):539–547.

- [71] Fachkha, C., Bou-Harb, E., and Debbabi, M. (2013). Towards a forecasting model for distributed denial of service activities. In *2013 IEEE 12th International Symposium on Network Computing and Applications*, pages 110–117.
- [72] Fakhruzi, I. (2018). An artificial neural network with bagging to address imbalance datasets on clinical prediction. In *2018 International Conference on Information and Communications Technology (ICOIACT)*, pages 895–898.
- [73] Fesin, P. and Anu, S. (2016). Survey on applications of ambient intelligence. *Journal of Artificial Intelligence Research Advances*, 3(2):44–48.
- [74] Fette, I. and Melnikov, A. (2011). The websocket protocol.
- [75] Font, J. L., Iñigo, P., Domínguez, M., Sevillano, J. L., and Amaya, C. (2011). Analysis of source code metrics from ns-2 and ns-3 network simulators. *Simulation Modelling Practice and Theory*, 19(5):1330–1346.
- [76] Foreman, N. and Evett, M. (2005). Preventing overfitting in gp with canary functions. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, GECCO '05*, page 1779–1780, New York, NY, USA. Association for Computing Machinery.
- [77] Foundation, F. (2018). What is chaos theory?
- [78] G. R. Clarke, D. R. and Wright, E. (2004). *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes.
- [79] Galloopeni, G., Rodrigues, B., Franco, M., and Stiller, B. (2020). A practical analysis on mirai botnet traffic. In *2020 IFIP Networking Conference (Networking)*, pages 667–668.
- [80] Gao, W., Sun, Y., Fu, Q., Wu, Z., Ma, X., Zheng, K., and Huang, X. (2018). Arp poisoning prevention in internet of things. In *2018 9th International Conference on Information Technology in Medicine and Education (ITME)*, pages 733–736.
- [81] Gaurav, A. and Singh, A. K. (2017). Entropy-score: A method to detect ddos attack and flash crowd. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 1427–1431.
- [82] Gaurav, A. and Singh, A. K. (2017). Entropy-score: A method to detect ddos attack and flash crowd. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 1427–1431.
- [83] Ge, M., Fu, X., Syed, N., Baig, Z., Teo, G., and Robles-Kelly, A. (2019). Deep learning-based intrusion detection for iot networks. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 256–25609.
- [84] Geissler, S., Wamser, F., Bauer, W., Krolikowski, M., Gebert, S., and Hoßfeld, T. (2021). Signaling traffic in internet-of-things mobile networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 452–458.
- [85] Girnar, N. and Kaur, S. (2017). Intrusion detection for adhoc networks in iot. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 110–114.

- [86] Gopstein, A., Nguyen, C., O'Fallon, C., Hastings, N., and Wollman, D. (2021). Nist framework and roadmap for smart grid interoperability standards, release 4.0.
- [87] Green, R. C., Wang, L., and Alam, M. (2013). Applications and trends of high performance computing for electric power systems: Focusing on smart grid. *IEEE Transactions on Smart Grid*, 4(2):922–931.
- [88] Greiner, W. (2010). *Lyapunov Exponents and Chaos*. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [89] Grochocki, D., Huh, J. H., Berthier, R., Bobba, R., Sanders, W. H., Cárdenas, A. A., and Jetcheva, J. G. (2012). Ami threats, intrusion detection requirements and deployment recommendations. In *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pages 395–400.
- [90] Grochocki, D., Huh, J. H., Berthier, R., Bobba, R., Sanders, W. H., Cárdenas, A. A., and Jetcheva, J. G. (2012). Ami threats, intrusion detection requirements and deployment recommendations. In *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pages 395–400.
- [91] H, A. Abdul-Ghani, D. K. and Mahyoub, M. (2018). A comprehensive iot attacks survey based on a building-blocked reference model. *International Journal of Advanced Computer Science and Applications*, 9(3).
- [92] H. H. Jazi, H. Gonzalez, N. S. and Ghorbani, A. A. (2017). Detecting http-based application layer dos attacks on web servers in the presence of sampling. *Computer Networks*, 121(C):25–36.
- [93] Haddad, I. and Gordon, D. (2002). Network simulator 2: a simulation tool for linux. *Linux Journal*, 10.
- [94] Hahm, O., Baccelli, E., Petersen, H., and Tsiftes, N. (2016). Operating systems for low-end devices in the internet of things: A survey. *IEEE Internet of Things Journal*, 3(5):720–734.
- [95] Harmon, R. R., Castro-Leon, E. G., and Bhide, S. (2015). Smart cities and the internet of things. In *2015 Portland International Conference on Management of Engineering and Technology (PICMET)*, pages 485–494.
- [96] Harper, R. (2006). *Inside the smart home*. Springer Science & Business Media.
- [97] He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284.
- [98] Hernandez, G., Arias, O., Buentello, D., and Jin, Y. (2014). Smart nest thermostat: A smart spy in your home. *Black Hat USA*, (2015).
- [99] Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P., Iorkyase, E., Tachtatzis, C., and Atkinson, R. (2016). Threat analysis of iot networks using artificial neural network intrusion detection system. In *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6.

- [100] Hoerl, A. E. and Kennard, R. W. (1970a). Ridge regression: Applications to nonorthogonal problems. *Technometrics*, 12(1):69–82.
- [101] Hoerl, A. E. and Kennard, R. W. (1970b). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.
- [102] Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10.
- [103] Huong, T. T., Bac, T. P., Long, D. M., Thang, B. D., Binh, N. T., Luong, T. D., and Phuc, T. K. (2021). Lockedge: Low-complexity cyberattack detection in iot edge computing. *IEEE Access*, 9:29696–29710.
- [104] Hwang, R., Peng, M., Huang, C., Lin, P., and Nguyen, V. (2020). An unsupervised deep learning model for early network traffic anomaly detection. *IEEE Access*, 8:30387–30399.
- [105] I. Sharafaldin, A. H. L. and Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *4th International Conference on Information Systems Security and Privacy (ICISSP) Portugal*, 1:108–116.
- [106] Ibitoye, O., Shafiq, O., and Matrawy, A. (2019). Analyzing adversarial attacks against deep learning for intrusion detection in iot networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.
- [107] Ioannou, C., Vassiliou, V., and Sergiou, C. (2017). An intrusion detection system for wireless sensor networks. In *2017 24th International Conference on Telecommunications (ICT)*, pages 1–5.
- [108] Jahromi, A. N., Karimipour, H., Dehghantanha, A., and Parizi, R. M. (2021). *Deep Representation Learning for Cyber-Attack Detection in Industrial IoT*, pages 139–162. Springer International Publishing, Cham.
- [109] Jaiganesh, V., Mangayarkarasi, S., and Sumathi, P. (2013). Intrusion detection systems: A survey and analysis of classification techniques. volume 2, pages 1629–1635.
- [110] Jiang, J., Jing, X., Lv, B., and Li, M. (2015). A novel multi-classification intrusion detection model based on relevance vector machine. In *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pages 303–307.
- [111] Jimoh, I. A., Ismaila, I., and Olalere, M. (2019). Enhanced decision tree-j48 with smote machine learning algorithm for effective botnet detection in imbalance dataset. In *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1–8.
- [112] Kahla, M., Azab, M., and Mansour, A. (2018). Secure, resilient, and self-configuring fog architecture for untrustworthy iot environments. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 49–54.

- [113] Kalekar, P. (2004). Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi School of Information Technology*, 4329008:1–13.
- [114] Kamilaris, A., Pitsillides, A., and Yiallourous, M. (2013). Building energy-aware smart homes using web technologies. *J. Ambient Intell. Smart Environ.*, 5(2):161–186.
- [115] Karim, L., Bensaber Boucif, A., Mhamed, M., and Ismail, B. (2015). A probabilistic model to corroborate three attacks in vehicular ad hoc networks. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 70–75.
- [116] Kawamura, T., Fukushi, M., Hirano, Y., Fujita, Y., and Hamamoto, Y. (2017). An ntp-based detection module for ddos attacks on iot. In *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pages 15–16.
- [117] Kempton, W. and Layne, L. (1994). The consumer’s energy analysis environment. *Energy Policy*, 22(10):857–866.
- [118] Khan, A. A. and Mouftah, H. T. (2011). Web services for indoor energy management in a smart grid environment. In *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1036–1040.
- [119] Kheaksong, A. and Lee, W. (2014). Packet transfer of dlms/cosem standards for smart grid. In *The 20th Asia-Pacific Conference on Communication (APCC2014)*, pages 391–396.
- [120] Kim, D.-S., Baek, Y.-M., and Kim, W.-Y. (2013). Reducing overfitting of adaboost by clustering-based pruning of hard examples. *ICUIMC ’13*, New York, NY, USA. Association for Computing Machinery.
- [121] Kloibhofer, S. (2017). *CIREN - Open Access Proceedings Journal*, 2017:1638–1641(3).
- [122] Koch, R., Kühn, T., Odenwald, M., and Dreö Rodosek, G. (2016). Dr. wattson: Lightweight current-based intrusion detection (cbid). In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 170–177.
- [123] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI’95*, page 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [124] Kohavi, R. (1996). *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. PhD thesis, Stanford, CA, USA. UMI Order No. GAX96-11989.
- [125] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324. Relevance.
- [126] Kolias, C., Kambourakis, G., Stavrou, A., and Voas, J. (2017). Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84.
- [127] Kubat, M., Holte, R. C., and Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Mach. Learn.*, 30(2–3):195–215.

- [128] Kulkarni, G., Shelke, R., Sutar, R., and Mohite, S. (2014). “rfid security issues and challenges”. In *2014 International Conference on Electronics and Communication Systems (ICECS)*, pages 1–4.
- [129] Lai, Y.-C. and Ye, N. (2003). Recent developments in chaotic time series analysis. *International Journal of Bifurcation and Chaos*, 13(6):1383–1422.
- [130] Laner, M., Svoboda, P., Nikaein, N., and Rupp, M. (2013). Traffic models for machine type communications. In *ISWCS 2013; The Tenth International Symposium on Wireless Communication Systems*, pages 1–5.
- [131] Leach, P. J., Berners-Lee, T., Mogul, J. C., Masinter, L., Fielding, R. T., and Gettys, J. (1999). Rfc 2616: Hypertext transfer protocol–http/1.1. *Network Working Group*.
- [132] Lee, C., Zappaterra, L., Choi, K., and Choi, H.-A. (2014). Securing smart home: Technologies, security challenges, and security requirements. In *2014 IEEE Conference on Communications and Network Security*, pages 67–72.
- [133] Lee, C., Zappaterra, L., Kwanghee Choi, and Hyeong-Ah Choi (2014). Securing smart home: Technologies, security challenges, and security requirements. In *2014 IEEE Conference on Communications and Network Security*, pages 67–72.
- [134] Lee, L.-M. and Lee, J.-C. (2006). A study on high-order hidden markov models and applications to speech recognition. In *Proceedings of the 19th International Conference on Advances in Applied Artificial Intelligence: Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE’06*, page 682–690, Berlin, Heidelberg. Springer-Verlag.
- [135] Legendre, A. M. (2018). *Nouvelles Méthodes Pour La Détermination Des Orbites Des Comètes*. WENTWORTH Press.
- [136] Li, H., Chen, Y., and He, Z. (2012). The survey of rfid attacks and defenses. In *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4.
- [137] Li, H., Li, J., Guan, X., Liang, B., Lai, Y., and Luo, X. (2019). Research on overfitting of deep learning. In *2019 15th International Conference on Computational Intelligence and Security (CIS)*, pages 78–81.
- [138] Li, K., Zhou, W., Li, P., Hai, J., and Liu, J. (2009a). Distinguishing ddos attacks from flash crowds using probability metrics. In *2009 Third International Conference on Network and System Security*, pages 9–17.
- [139] Li, K., Zhou, W., Li, P., Hai, J., and Liu, J. (2009b). Distinguishing ddos attacks from flash crowds using probability metrics. In *2009 Third International Conference on Network and System Security*, pages 9–17.
- [140] Liu, J., Xiao, Y., Li, S., Liang, W., and Chen, C. L. P. (2012). Cyber security and privacy issues in smart grids. *IEEE Communications Surveys Tutorials*, 14(4):981–997.
- [141] Liu, J., Xiao, Y., Li, S., Liang, W., and Chen, C. L. P. (2012). Cyber security and privacy issues in smart grids. *IEEE Communications Surveys Tutorials*, 14(4):981–997.

- [142] Lorenz, E. (1979). Predictability: Does the flap of a butterfly wings in brazil set of a tornado in texas. *Proceedings American Association for the Advancement of Science in Washington*.
- [143] Lu, Y., Wu, F., Tang, S., Kong, L., and Chen, G. (2019). Pushing the limit of csi-based activity recognition: An enhanced approach via packet reconstruction. In *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9.
- [144] M. Hall, E. Frank, G. H. B. P. P. R. and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.
- [145] M. J. Schwartz (2013). Bank attackers restart operation ababil ddos disruptions. [online] <https://www.darkreading.com/attacks-and-breaches/bank-attackers-restart-operation-ababil-ddos-disruptions/d/d-id/1108955>.
- [146] Ma, X. and Chen, Y. (2014). Ddos detection method based on chaos analysis of network traffic entropy. *IEEE Communications Letters*, 18(1):114–117.
- [147] Ma, X. and Chen, Y. (2014). Ddos detection method based on chaos analysis of network traffic entropy. *IEEE Communications Letters*, 18(1):114–117.
- [148] Machaka, P., Bagula, A., and Nelwamondo, F. (2016). Using exponentially weighted moving average algorithm to defend against ddos attacks. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pages 1–6.
- [149] Mackiewicz, R. E. (2006). Overview of iec 61850 and benefits. In *2006 IEEE PES Power Systems Conference and Exposition*, pages 623–630.
- [150] Malialis, K. (2014). Distributed reinforcement learning for network intrusion response.
- [151] Malialis, K., Devlin, S., and Kudenko, D. (2015). Distributed reinforcement learning for adaptive and robust network intrusion response. *Connection Science*, 27(3):234–252.
- [152] Manimurugan, S., Al-Mutairi, S., Aborokbah, M. M., Chilamkurti, N., Ganesan, S., and Patan, R. (2020). Effective attack detection in internet of medical things smart environment using a deep belief neural network. *IEEE Access*, 8:77396–77404.
- [153] Marquardt, D. W. and Snee, R. D. (1975). Ridge regression in practice. *The American Statistician*, 29(1):3–20.
- [154] McDowell, M. (2004). Understanding denial-of-service attacks. *National Cyber Alert System, Cyber Security Tip ST04-015.2004*.
- [155] Medhat, K., Ramadan, R. A., and Talkhan, I. (2015). Distributed intrusion detection system for wireless sensor networks. In *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 234–239.
- [156] Mentzer, J. T. and Gomes, R. (1994). Further extensions of adaptive extended exponential smoothing and comparison with the m-competition. *Journal of the Academy of Marketing Science*, 22(4):372–382.

- [157] Metzger, F., Hoßfeld, T., Bauer, A., Kounev, S., and Heegaard, P. E. (2019). Modeling of aggregated iot traffic and its application to an iot cloud. *Proceedings of the IEEE*, 107(4):679–694.
- [158] Mirkovic, J. and Reiher, P. (2004a). A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53.
- [159] Mirkovic, J. and Reiher, P. (2004b). A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34.
- [160] Mitchell, T. M. (2006). *The discipline of machine learning*, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning Department Pittsburgh, PA.
- [161] M.M Salim, S. R. and Park., J. (2020). Distributed denial of service attacks and its defenses in iot: a survey. *The Journal of Supercomputing*, 20(2):5320–5363.
- [162] Modarres, Z. G., Shabankhah, M., and Kamandi, A. (2020). Making adaboost less prone to overfitting on noisy datasets. In *2020 6th International Conference on Web Research (ICWR)*, pages 251–259.
- [163] Mohsen, H. (2012). A study of the source traffic generator using poisson distribution for abr service. *Modelling and Simulation in Engineering*, 2012.
- [164] More, A. (2016). Survey of resampling techniques for improving classification performance in unbalanced datasets. *ArXiv*, abs/1608.06048.
- [165] Mosenia, A. and Jha, N. K. (2017). A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing*, 5(4):586–602.
- [166] Moser, K., Harder, J., and Koo, S. G. M. (2014). Internet of things in home automation and energy efficient smart home technologies. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1260–1265.
- [167] Moustis, D. and Kotzanikolaou, P. (2013). Evaluating security controls against http-based ddos attacks. In *IISA 2013*, pages 1–6.
- [168] Mudgerikar, A., Sharma, P., and Bertino, E. (2020). Edge-based intrusion detection for iot devices. *ACM Trans. Manage. Inf. Syst.*, 11(4).
- [169] N. Ye, Q. C. and Borror, C. M. (2004). Ewma forecast of normal system activity for computer intrusion detection. *IEEE Transactions on Reliability*, 53(4):557–566.
- [170] N. Koroniotis, M. Nour, E. S. and Turnbull, B. (2019). Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796.
- [171] Najafabadi, M. M., Khoshgoftaar, T. M., Calvert, C., and Kemp, C. (2017). User behavior anomaly detection for application layer ddos attacks. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 154–161.
- [172] Namboodiri, V., Aravinthan, V., Mohapatra, S. N., Karimi, B., and Jewell, W. (2014). Toward a secure wireless-based home area network for metering in smart grids. *IEEE Systems Journal*, 8(2):509–520.

- [173] Namboodiri, V., Aravinthan, V., Mohapatra, S. N., Karimi, B., and Jewell, W. (2014). Toward a secure wireless-based home area network for metering in smart grids. *IEEE Systems Journal*, 8(2):509–520.
- [174] P. Paganini (2015). Protonmail paid a 6000 usd ransom to stop ddos attacks. [online] <https://www.securityaffairs.com/wordpress/41775/cyber-crime/protonmail-paid-ransom-ddos.html>.
- [175] P. Wendell, P. and Freedman, M. J. (2011). Going viral: Flash crowds in an open cdn. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, page 549–558, New York, NY, USA. Association for Computing Machinery.
- [176] P. Winter, H. Lampesberger, M. Z. and Hermann, E. (2011). On detecting abrupt changes in network entropy time series. In De Decker, B., Lapon, J., Naessens, V., and Uhl, A., editors, *Communications and Multimedia Security*, pages 194–205, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [177] Pajouh, H. H., Javidan, R., Khayami, R., Dehghantanha, A., and Choo, K. R. (2019). A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in iot backbone networks. *IEEE Transactions on Emerging Topics in Computing*, 7(2):314–323.
- [178] Papadie, R. and Apostol, I. (2017). Analyzing websites protection mechanisms against ddos attacks. In *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6.
- [179] Park, J., Chebbah, M., Jendoubi, S., and Martin, A. (2014). Second-order belief hidden markov models. In Cuzzolin, F., editor, *Belief Functions: Theory and Applications*, pages 284–293, Cham. Springer International Publishing.
- [180] Pongle, P. and Chavan, G. (2015). A survey: Attacks on rpl and 6lowpan in iot. In *2015 International Conference on Pervasive Computing (ICPC)*, pages 1–6.
- [181] Prabavathy, S., Sundarakantham, K., and Shalinie, S. M. (2018). Design of cognitive fog computing for intrusion detection in internet of things. *Journal of Communications and Networks*, 20(3):291–298.
- [182] Puri, A. and Gupta, M. K. (2019). Comparative analysis of resampling techniques under noisy imbalanced datasets. In *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, volume 1, pages 1–5.
- [183] Qian, L., Hu, L., Zhao, L., Wang, T., and Jiang, R. (2020). Sequence-dropout block for reducing overfitting problem in image classification. *IEEE Access*, 8:62830–62840.
- [184] R. A. Shah, Y. Qian, D. K. M. A. and Alvi, M. B. (2017). Network intrusion detection through discriminative feature selection by using sparse logistic regression. *Future Internet*, 9(4).
- [185] R. Dunia, R. and Qin, J. (1997). Multi-dimensional fault diagnosis using a subspace approach.

- [186] R.A. Shah, Y. Qian, D. K. M. A. and Alvi, M. (2017). Network intrusion detection through discriminative feature selection by using sparse logistic regression. *Future Internet*, 9(4).
- [187] Rajeswari, L. P. and Kannan, A. (2008). An intrusion detection system based on multiple level hybrid classifier using enhanced c4.5. In *2008 International Conference on Signal Processing, Communications and Networking*, pages 75–79.
- [188] Rani, T. S., Sindhura, V., Rao, G. R., and Pranathi, K. (2015). Discerning flooding attack from flash crowd based on traffic patterns using entropy detection method. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pages 1–6.
- [189] Ranjith (2018). Goldeneye – goldeneye layer 7 dos test tool. KaliLinuxTutorials. [online] Available: <https://kalilinuxtutorials.com/goldeneye-dos-test-tool/>.
- [190] Raudys, S. and Jain, A. (1991). Small sample size effects in statistical pattern recognition: recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):252–264.
- [191] Ravi, N. and Shalinie, S. M. (2020). Learning-driven detection and mitigation of ddos attack in iot via sdn-cloud architecture. *IEEE Internet of Things Journal*, 7(4):3559–3570.
- [192] Rej, R. (2003). Nist/sematech e-handbook of statistical methods. *Clinical Chemistry*, 49(6):1033–1034.
- [193] Roohi, A., Adeel, M., and Shah, M. A. (2019). Ddos in iot: A roadmap towards security countermeasures. In *2019 25th International Conference on Automation and Computing (ICAC)*, pages 1–6.
- [194] Roopak, M., Tian, G. Y., and Chambers, J. (2020). An intrusion detection system against ddos attacks in iot networks. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0562–0567.
- [195] S. De Mata (2015). Ns-3-http-traffic-generator. [online] <https://github.com/saulodamata/ns-3-http-traffic-generator>.
- [196] S. Shekyan (2016). slowhttpptest. Github. [online] Available: <https://github.com/shekyan/slowhttpptest/wiki>.
- [197] Saint-Andre, P. (2011). Extensible messaging and presence protocol (xmpp): address format. Technical report, RFC 6122, March.
- [198] Samy, A., Yu, H., and Zhang, H. (2020). Fog-based attack detection framework for internet of things using deep learning. *IEEE Access*, 8:74571–74585.
- [199] Santoyo-González, A., Cervelló-Pastor, C., and Pezaros, D. P. (2019). High-performance, platform-independent ddos detection for iot ecosystems. In *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, pages 69–75.
- [200] Sarma, D., Alam, W., Saha, I., Alam, M. N., Alam, M. J., and Hossain, S. (2020). Bank fraud detection using community detection algorithm. In *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 642–646.

- [201] Shahid, M. R., Blanc, G., Zhang, Z., and Debar, H. (2018). Iot devices recognition through network traffic analysis. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5187–5192.
- [202] Shorey, T., Subbaiah, D., Goyal, A., Sakxena, A., and Mishra, A. K. (2018). Performance comparison and analysis of slowloris, goldeneye and xerxes ddos attack tools. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 318–322.
- [203] Shukla, P. (2017). MI-ids: A machine learning approach to detect wormhole attacks in internet of things. In *2017 Intelligent Systems Conference (IntelliSys)*, pages 234–240.
- [204] Siano, P. (2014). Demand response and smart grids—a survey. *Renewable and Sustainable Energy Reviews*, 30:461 – 478.
- [205] Sikora, M., Gerlich, T., and Malina, L. (2019). On detection and mitigation of slow rate denial of service attacks. In *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 1–5.
- [206] Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. (2019). Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759.
- [207] Sivanathan, A., Sherratt, D., Gharakheili, H. H., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. (2017). Characterizing and classifying iot traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564.
- [208] SKK University (2015). Ns-3 iotivity module, 2015. [online] <https://www.comnnet.skku.edu>.
- [209] Skopik, F. and Ma, Z. (2012). Attack vectors to metering data in smart grids under security constraints. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, pages 134–139.
- [210] Skorokhod, A. (2002). *Foundations of Modern Probability*. Springer Science and Business Media.
- [211] Skorokhod, A. (2005). *Basic Principles and Applications of Probability Theory*. Springer Science and Business Media.
- [212] Song, J., Huang, X., Qin, S., and Song, Q. (2016). A bi-directional sampling based on k-means method for imbalance text classification. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–5.
- [213] Statsoft(firm) (1997). *Electronic statistics textbook*. Tulsa Publishers.
- [214] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147.
- [215] Stone, M. (1977). An asymptotic equivalence of choice of model by cross-validation and akaike’s criterion. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):44–47.

- [216] Subba, B., Biswas, S., and Karmakar, S. (2015). Intrusion detection systems using linear discriminant analysis and logistic regression. In *2015 Annual IEEE India Conference (INDICON)*, pages 1–6.
- [217] Swaroopa Rani, T., Sindhura, V., RamaKoteswara Rao, G., and Pranathi, K. (2015). Discerning flooding attack from flash crowd based on traffic patterns using entropy detection method. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–6.
- [218] Tabatabaie Nezhad, S. M., Nazari, M., and Gharavol, E. A. (2016). A novel dos and ddos attacks detection algorithm using arima time series model and chaotic system in computer networks. *IEEE Communications Letters*, 20(4):700–703.
- [219] Tahaei, H., Affi, F., Asemi, A., Zaki, F., and Anuar, N. B. (2020). The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications*, 154:102538.
- [220] Taylor, J. W. (2004). Smooth transition exponential smoothing. *Journal of Forecasting*, 23(6):385–404.
- [221] Tchakoucht, T. A. and Ezziyyani, M. (2018). Detecting network intrusions using multi-class logistic regression and correlation-based feature selection. In *Advanced Information Technology, Services and Systems*, pages 416–426, Cham. Springer International Publishing.
- [222] Thapngam, T., Yu, S., Zhou, W., and Beliakov, G. (2011). Discriminating ddos attack traffic from flash crowd through packet arrival patterns. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 952–957.
- [223] U. A. Sandhu, S. Haider, S. N. and O. U. Ateeb, O. (2011). A survey of intrusion detection & prevention techniques. In *2011 International Conference on Information Communication and Management, IPCSIT*, volume 16, pages 66–67.
- [224] Vaidya, B., Makrakis, D., and Mouftah, H. (2012). Secure remote access to smart energy home area networks. In *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, pages 1–7.
- [225] Van Der Voort, M., Dougherty, M., and Watson, S. (1996). Combining kohonen maps with arima time series models to forecast traffic flow. *Transportation Research Part C: Emerging Technologies*, 4(5):307–318.
- [226] Varga, A. and Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems and Workshops, Simutools '08*, Brussels, BEL. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [227] Vinoski, S. (2006). Advanced message queuing protocol. *IEEE Internet Computing*, 10(6):87–89.
- [228] Vyas, D. (2012). Advance metering infrastructure and dlms/cosem standards for smart grid. *International Journal of Engineering Research Technology (IJERT)*, 1.

- [229] Žegklitz, J. and Pošík, P. (2015). Model selection and overfitting in genetic programming: Empirical study. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, page 1527–1528, New York, NY, USA. Association for Computing Machinery.
- [230] Wang, Y., Liu, L., Si, C., and Sun, B. (2017). A novel approach for countering application layer ddos attacks. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1814–1817.
- [231] Wankhade, A. and Chandrasekaran, K. (2016). Distributed-intrusion detection system using combination of ant colony optimization (aco) and support vector machine (svm). In *2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, pages 646–651.
- [232] Wheelus, C., Bou-Harb, E., and Zhu, X. (2018). Tackling class imbalance in cyber security datasets. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 229–232.
- [233] Winter, P., Lampesberger, H., Zeilinger, M., and Hermann, E. (2011). On detecting abrupt changes in network entropy time series. volume 7025.
- [234] Wolf, A., Swift, J. B., Swinney, H. L., and Vastano, J. A. (1985). Determining lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317.
- [235] Wolfe, J., Jin, X., Bahr, T., and Holzer, N. (2017). Application of softmax regression and its validation for spectral-based land cover mapping. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:455.
- [236] Wu, D., Yan, J., Wang, H., and Wang, R. (2019). Multiattack intrusion detection algorithm for edge-assisted internet of things. In *2019 IEEE International Conference on Industrial Internet (ICII)*, pages 210–218.
- [237] Wu, H. and Shapiro, J. L. (2006). Does overfitting affect performance in estimation of distribution algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 433–434, New York, NY, USA. Association for Computing Machinery.
- [238] Wu, X. and Chen, Y. (2013). Validation of chaos hypothesis in nada and improved ddos detection algorithm. *IEEE Communications Letters*, 17(12):2396–2399.
- [239] Wu, X. and Chen, Y. (2013). Validation of chaos hypothesis in nada and improved ddos detection algorithm. *IEEE Communications Letters*, 17(12):2396–2399.
- [240] Xie, Y. and Yu, S. (2009a). A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Transactions on Networking*, 17(1):54–65.
- [241] Xie, Y. and Yu, S. (2009b). Monitoring the application-layer ddos attacks for popular websites. *IEEE/ACM Transactions on Networking*, 17(1):15–25.

- [242] Xiong, W., Hu, H., Yang, Y., and Wang, Q. (2010). Anomaly detection of network traffic based on the largest lyapunov exponent. In *2010 2nd International Conference on Advanced Computer Control*, volume 4, pages 581–585.
- [243] Yadav, S. and Selvakumar, S. (2015). Detection of application layer ddos attack by modeling user behavior using logistic regression. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, pages 1–6.
- [244] Yadav, S. and Subramanian, S. (2016a). Detection of application layer ddos attack by feature learning using stacked autoencoder. In *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, pages 361–366.
- [245] Yadav, S. and Subramanian, S. (2016b). Detection of application layer ddos attack by feature learning using stacked autoencoder. In *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, pages 361–366.
- [246] Yan, Q., Jiancheng, Y., Xiangyu, K., Xudong, W., Yuying, M., and Bo, L. (2018). Home smart energy network configuration considerations for demand response. In *2018 China International Conference on Electricity Distribution (CICED)*, pages 2875–2879.
- [247] Yan, R., Xu, G., and Qin, X. (2017). Detect and identify ddos attacks from flash crowd based on self-similarity and renyi entropy. In *2017 Chinese Automation Congress (CAC)*, pages 7188–7194.
- [248] Yan, R., Xu, G., and Qin, X. (2017). Detect and identify ddos attacks from flash crowd based on self-similarity and renyi entropy. In *2017 Chinese Automation Congress (CAC)*, pages 7188–7194.
- [249] Yan, Y., Qian, Y., Sharif, H., and Tipper, D. (2013). A survey on smart grid communication infrastructures: Motivations, requirements and challenges. *IEEE Communications Surveys Tutorials*, 15(1):5–20.
- [250] Ye, C., Zheng, K., and She, C. (2012). Application layer ddos detection using clustering analysis. In *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, pages 1038–1041.
- [251] Yu, S., Zhou, W., Jia, W., Guo, S., Xiang, Y., and Tang, F. (2012). Discriminating ddos attacks from flash crowds using flow correlation coefficient. *IEEE Transactions on Parallel and Distributed Systems*, 23(6):1073–1080.
- [252] Yu, S., Zhou, W., Jia, W., Guo, S., Xiang, Y., and Tang, F. (2012). Discriminating ddos attacks from flash crowds using flow correlation coefficient. *IEEE Transactions on Parallel and Distributed Systems*, 23(6):1073–1080.
- [253] Z. Banks (2019). Slowloris http denial of service. Hackaday. [online]Available: <https://hackaday.com/2009/06/17/slowloris-http-denial-of-service/>.

- [254] Zamini, M. and Montazer, G. (2018). Credit card fraud detection using autoencoder based clustering. In *2018 9th International Symposium on Telecommunications (IST)*, pages 486–491.
- [255] Zhang, Y., Wang, L., Sun, W., Green II, R. C., and Alam, M. (2011). Distributed intrusion detection system in a multi-layer network architecture of smart grids. *IEEE Transactions on Smart Grid*, 2(4):796–808.
- [256] Zhao, S., Li, W., Zia, T., and Zomaya, A. Y. (2017). A dimension reduction model and classifier for anomaly-based intrusion detection in internet of things. In *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 836–843.

