Degree in Computer engineering
Computer Engineering

Final degree project

Detection of network security attacks in real time

Author

Iker Foronda Carrasco

2020

eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Degree in Computer engineering

Computer Engineering

Final degree project

Detection of network security attacks in real time

Author

Iker Foronda Carrasco

Directors

Jose A. Pascual Saiz and Roberto Santana Hermida

Abstract

As technology advances towards progress and innovation, its importance has increased exponentially as the years have passed, mainly in regard of management tasks such as finances, employment, analysis or marketing. Not only has this importance increased in the scope of business interactions, but it has also changed the way people interact forever. Social networks, videogames and entertainment are a clear example of this. In a context where all the computation is starting to become more and more distributed, the computer networks also become critical financial assets for the enterprises.

Computer networks have become the target of several attacks. The goal of those attacks is to disrupt the normal behaviour of the services provided from inside the network (DoS attacks) or to gain access to the systems offering those services. For that reason, the detection of those attacks as soon as possible is key to maintain the security of the whole network. This is precisely the objective of this project: the development of a detection module together with a fast network analyzer which will allow detecting such attacks in real-time. The tools that will be used to fulfill the goal are machine learning approaches and the labeled dataset UNSW-NB15 which contains a lot of different types of attacks.

Contents

List of Figures

List of Tables

1. CHAPTER

Introduction

This project addresses two leading-edge topics of the computer science field, which have attracted both the attention of the media and the interest of the researchers, as there are still many discoveries to be done. In fact, the potential of these branches of knowledge is proven to be great, as this branches can be used in order to create network security systems, which is the aim of this project, the development of a NIDS.

This chapter is an introduction to the topics of machine learning and computer networks, in order to analyze which concepts can be applied to the network security aspect. The objective of this chapter is to give a general taste and contextualization on the topics addressed more deeply during the document.

1.1 Computer Networks and security

Cybersecurity includes the management of the elements which must be protected from the threats associated with them using some mechanisms, in order to preserve the security services associated with them. The main aim of the cybersecurity is ensure the reliability of these services, in the scope of this project, the focus is network security, the subset of security measures related to the computer networks.

Computer networking corresponds to the term used for the definition of a set of computers which are interconnected and communicate among themselves with the aid of communication protocols, over wired or wireless infrastructures. These are the common elements which can be identified as general ones, even though a more fine-grained distinction can be made. Although it has been mentioned these networks are groups of computers, they are in fact a mixture of servers, clients and networking hardware.

As the objective of this project is closely related to computer security, it is worth mentioning this branch of knowledge focuses on the implementation and assurance of secure services and communications, which assumes that computer security involves the means used in order to fulfill those objectives. These means include functions, features or technical information of the hardware or software.

The elements involved in this scope are the hardware, software and the data, whereas the threats could be divided by their origin, as a threat could be anything, a person, a program or even a catastrophe, and this division can be made also by the type of attack which is being performed.

The mechanisms try to protect the services on any of the stages a threat could appear. Therefore, there are prevention measures, detection measures, reactive measures and recovery measures.

The security services are being redefined constantly as the cybersecurity includes more elements to protect and the industry diversifies. However, the main security services are the following:

- **Confidentiality:** Service which provides the information the protection to not to be revealed to any excluding the entity which is supposed to know the information.
- **Integrity:** Service which ensures the information has not been adulterated, duplicated or deleted without permission or accidentally.
- **Authentication:** Service which ensures the identity of the entities and the origin of the information.
- **Access control:** Service which ensures the access to the resources only to the entities meant for that access.
- **Availability:** Service which ensures a resource is accessible and usable when requested.
- **Anonymity and Privacy:** Recently discussed security services, which aim is to hide the identities and activities of the entities involved in a communication.

The networks can be categorized between private or public, depending on their visibility, as the private ones should not be accessed from the "exterior", as they are aimed to be used only by authorized people and machines. Moreover, the security on these focuses to ensure the availability of the services and the access to privileged information, whereas public networks are much bigger in comparison, therefore the objective is to ensure the confidentiality of the message mainly.

1.2 Machine Learning

The project is going to use an approach based on some algorithms. The algorithms are categorized as machine learning based approaches, being this a branch of knowledge which belongs to the broad scope of the artificial intelligence. The main distinction from the usual AI and ML is that ML "learns" by "experience" in order to perform better in the metric which designates the "performance" on the task to be addressed [Mitchell, 1997], which in practice is achieved by letting the algorithm build a mathematical representation of the data which is fed to the algorithm. This representation is usually referred to as "model". In this section, three suitable algorithms for the task are going to be presented.

1.2.1 Multi Layer Perceptron

The Multi Layer Perceptron (MLP) is a neural network which is conformed by smaller units, which are called perceptrons. A perceptron is an algorithm for supervised learning for a binary classification task. The function which a perceptron can model is linear, and it is mapped using the dot product of the input vector (x) and the weights vector (w). Bias (b) is also added, as it modifies slightly the outputs of the function, which makes the function fit better the data.

The capacity of one perceptron is limited, as it can model only linear functions. However, these perceptrons can be stacked in layers and then layers can be connected with each other, increasing the whole architectures' capacity, making it able to estimate more complex functions [Cybenko, 1989].

1.2.2 Random Forest

Random Forest (RF) is a machine learning method used for classification and regression tasks. The algorithm operates by generating a set of randomly defined decision trees and using all of them to compute the classification or regression given a sample of data, then all the results are evaluated by the algorithm, and the final result is output as the option that has been returned most often [Breiman, 2001].

1.2.3 K Nearest Neighbors

K Nearest Neighbors (KNN) is a machine learning algorithm which is used for supervised learning tasks. The main assumption this algorithm benefits from is the concept of proximity. Given each of the data samples can be represented as points considering all of the features, the algorithm bases its decision on the closeness between the point which must be evaluated and the other points which have already been learned. The algorithm calculates the distances of the points closer to it, and the K nearest points will be considered for the final prediction, which is computed using the labels associated with the K closest points, as the output will be the class with most representation from those K points [Altman, 1992].

1.3 Scope of the project

Having contextualized the project, the relation of each of the branches of knowledge with the project will be explained, in order to identify the benefits the developed modules give to each of the branches.

The project is related to the computer network area because the modules developed have a direct impact in the way computer networks could be arranged, as the module could be added as a node to an already established network, or it could even become a key element of the management of a network, allowing new arrangements of nodes.

The relation the modules have with the computer security is direct as well, as the module itself can be categorized as an NIDS, which is the acronym for Network Intrusion Detection System, an appliance or program which monitors the network traffic in order to identify threats, and it is used as a prevention measure in private networks, used mainly for access control, authentication anomalies and misuse of the resources of the network.

The machine learning algorithms are the tools used in order to solve the task, as they are the key element which will be used in order to develop the logic behind the NIDS.

2. CHAPTER

Project Objectives

The aim for this project is to develop and integrate two modules. The task which the first module will perform is related to the analysis of the traffic within the network, as the module extracts information from it and processes the information in order to extract the needed features in form of data samples, which are then sent to the other module.

The second module will be based on a machine learning algorithm, and the task associated to it is to learn a ML model in order to classify the status of the network. The goal is to distinguish between normal and abnormal behaviour, or even classify the attack which is being performed. The dataset used for training this model is the UNSW-NB15 dataset, which provides samples of network traffic data, in form of a 47 feature array. Two of these features are labels that correspond to the status of the network associated to that sample, and, in case it corresponds to an ongoing attack, the second label represents the type of attack.

The last part of the project corresponds to the integration of the two modules. The network model will be in charge of capturing and processing traffic from the network in an efficient way and to send this formatted information to the second module, that using a previously trained ML model using the dataset, will be able to determine the status of the networks and to classify the type of attack that is being carried out. In Figure 2.1 a representation of the whole system has been depicted.

The main difficulties that this structure presents are related to the gathering of the features themselves from the network, as these must be collected in real-time, as the aim of the model is to take these decisions as soon as possible. The processing time of the Net-

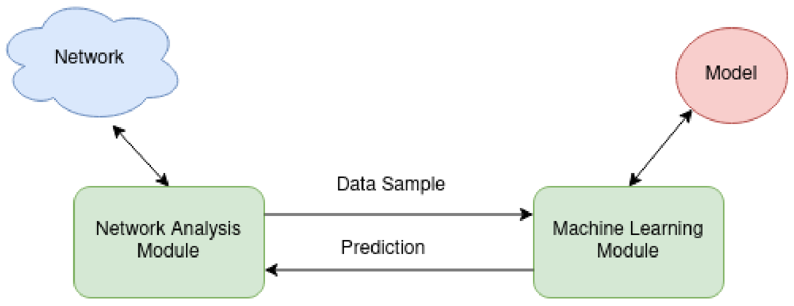


Figure 2.1: Conceptual representation of the system.

work Analysis module needs to be minimum, although the information extracted from the network must be adequate enough to fulfill the task.

In order to select which features must be extracted from the traffic, an analysis from the dataset must be carried first, in order to select the most relevant features for the model, and reduce the scope of features used for the prediction to only those features. The following steps include the definition and the implementation of the NAM, which will be implemented using C. The definition, training and testing of the machine learning model will be implemented in Python using libraries such as Keras or Sklearn. Once both modules work properly, the integration of both modules will be performed in order to achieve the desired goal.

The following list summarizes the tasks which have to be done in order to achieve the objective of the project:

1. **Analysis** of the dataset and **extraction** of the most remarkable features.
2. **Design** and **implementation** of the **Network Analysis module**.
3. **Design** and **implementation** of the **Machine Learning module**.
4. **Integration** of both modules.

3. CHAPTER

Analysis of the UNSW-NB15 dataset

This project is structured in a modular way, as its composed by two independent modules, which after their implementation will communicate with each other. As mentioned before, the objective is the detection of attacks to a computer network in real-time.

The first module is the *Network Analysis module* which captures and extracts information from a computer network. The inner workings of the module are simple, as the module is a sniffer which collects, extracts and analyzes the information from the packets traveling the network in real-time. After a certain amount of packets is gathered, that information is processed and transformed into a “sample-alike” structure, which is then passed to the Machine Learning module to be classified.

The second module is the ML module, which acts as the element of decision of the system making predictions about the actual security state of the network. This is carried out using a ML model which classifies samples of data provided by the NAM. The model will be trained using the UNSW-NB15 dataset which contains information (features) about known attacks to computer networks. The task to be solved is a supervised classification problem, as each sample is labeled with the actual status of the network during the extraction of the sample, and if it was under an attack, the class of attack that was being performed. Therefore, this detection of ongoing attacks can be addressed as a binary classification problem if the distinction is done between regular and malicious behaviour, or as a multi label classification problem if the class of attack must be identified.

For this case, some machine learning algorithms and techniques will be used in order to find the best approach to solve the task. The main metric to be used could be accuracy, as for a NIDS might be a good idea to solve the problem in terms of maximizing the correct prediction count. However, a NIDS with a high false negative count would be a catastrophic scenario, given that the accuracy of the model could be high, even though the model would ignore completely malicious traffic. Therefore, it is a sensible choice to use any metric which is more balanced, such as f1 score, as it seems more reasonable in terms of considering al the possible predictions given by the algorithm.

The number of features which can be extracted from the network traffic is huge. However, as the analysis module must process and create the samples in a narrow window of time, the amount of required processing must be reduced. This objective can be achieved in two ways: on the one hand, the processing power can be increased, which is a costly yet convenient solution, as it does not add any extra complexity to our task. On the other hand, the information to be gathered can be reduced in order to ease the processing the sniffer needs to generate a data sample. This task can be achieved using *feature selection* techniques [Kira and Rendell, 1992].

The approach taken in this project is the latter: the reduction of the amount of features used for training the model, with the aim of reducing the generation time of the samples which is critical for the performance of the system. The features to be chosen will be selected using state-of-the-art algorithms which will be explained later.

3.1 Description of the dataset

The dataset used to train and test the model used in the ML modules is the UNSW-NB15 dataset [Moustafa and Slay, 2015], generated by the ACCS (Australian Centre for Cyber Security), as an evaluation tool for NIDSs. This dataset is considered to be the improved version of other datasets with have the same objective, such as KDDCUP 99 [Tavallaee et al., 2009] and NSLKDD [Meena and Choudhary, 2017]. The UNSW-NB15 dataset tries to address and solve the problems found in those mentioned datasets, hence, it is the natural choice to train and test the model.

A NIDS dataset can be conceptualized as relational data, that is, a set of data records that serves as input to a NIDS. Each record consists of a number of attributes of different data types (e.g., binary, float, nominal and integer). In order to make the dataset usable by the machine learning algorithms, the data has to be preprocessed mapping the nominal and categorical features to numerical ones.

The dataset is synthetically generated in a controlled network, using the IXIA [1] traffic generator and Argus[2] and Bro-IDS [3] in order to extract the features from the pcap files. Additionally, deeper analysis is done when monitoring the flows, in order to extract complex features which give additional information about the network status. The environment where these tools interact is defined by three virtual servers, which generate the traffic of the network. Servers 1 and 3 generate standard, non-malicious traffic, while Server 2 generates abnormal, malicious traffic based on information received from a Common Vulnerabilities and Exposures (CVE) site, where many records of known attacks are stored.

The flows generated are a mixture of private and public traffic. The servers are connected to the hosts via two routers, which are connected to the firewall, configured in such a way it would let pass through any kind of traffic. The packet capturer is installed in router number 1.

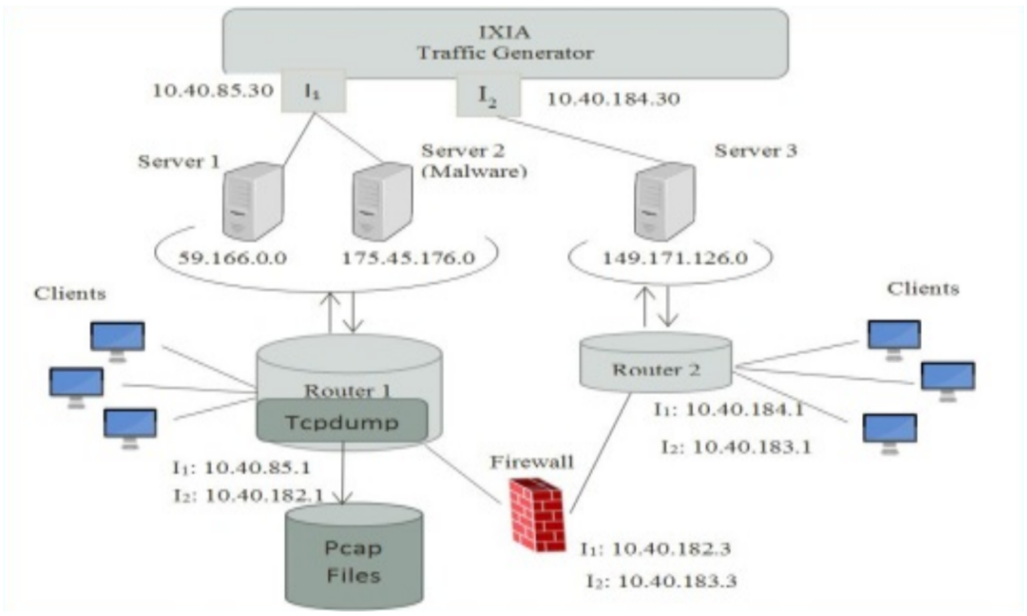


Figure 3.1: Data generation architecture used for the UNSW-NB15 dataset [4].

The whole architecture, depicted in Figure 3.1, is involved in generating the final shape of the UNSW-NB15 dataset following the next sequence: firstly, raw traffic is extracted from the network and stored in *pcap* files using Argus and Bro-IDS. These tools generate many

[1] https://www.ixiacom.com/solutions/network-test-solutions
[2] https://openargus.org/
[3] https://old.zeek.org/manual/2.5.5/index.htmltools
[4] https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/

simple features which are then converted into complex ones applying some algorithms. Once all the features have been generated, the sample is stored in a *CSV* file.

Each sample is formed by 47 features, however, some features such as the *ID*, which is an identifier for the sample order within the dataset, does not add any meaningful information for the task at hand and having it included in the dataset could act in detriment of the permutation importance calculation [Andre Altmann, 2010], therefore, it is removed. The same is done for IP addresses and ports, as they are flow-related features, and they are only used for extracting the complex features. In practice, the dataset is annotated with 42 features and two labels that classify each sample as normal/malicious behaviour. Malicious labels also include information about the type of attack.

The ordered feature list is shown below with the labels used in the dataset, where the index is the same as the one used for identification of the column in the *CSV* file:

0. **Duration:** The duration of the record.
1. **Protocol:** Transaction protocol.
2. **Service:** Type of service used (e.g., HTTP, DNS, FTP, etc).
3. **State:** Status of the dependent protocol.
4. **Spkts:** Packets sent from source to destination.
5. **Dpkts:** Packets sent from destination to source.
6. **Sbytes:** Bytes sent from source to destination.
7. **Dbytes:** Bytes sent from destination to source.
8. **Rate:** Communications packet latency (ms).
9. **STTL:** Source to destination Time-To-Live.
10. **DTTL:** Destination to source Time-To-Live.
11. **Sload:** Source bits per second.
12. **Dload:** Destination bits per second.
13. **Sloss:** Source packets that are retransmitted/dropped.
14. **Dloss:** Destination packets that are retransmitted/dropped.

15. **Sinpkt:** Source inter-arrival packet time (ms).
16. **Dinpkt:** Destination inter-arrival time (ms).
17. **Sjit:** Source jitter (ms).
18. **Djit:** Destination jitter (ms).
19. **Swin:** Source TCP window advertisement.
20. **Dwin:** Destination TCP window advertisement.
21. **Stcpb:** Source TCP sequence number.
22. **Dtcpb:** Destination TCP sequence number.
23. **TCPRTT:** The sum of SYNACK and ACKDAT feature values.
24. **SYNACK:** The time between the SYN and SYNACK packets on a TCP connection.
25. **ACKDAT:** The time between the SYNACK and ACK packets of the TCP connection.
26. **Smean:** Mean of the flows packet size transmitted by source.
27. **Dmean:** Mean of the flows packet size transmitted by destination
28. **Trans_depth:** Depth into the connection of a HTTP transaction.
29. **Response_body_len:** The content size of the data transferred from a HTTP service.
30. **Ct_srv_src:**Number of connections that contain the same service and source address in 100 connections according to the last time.
31. **Ct_state_TTL:**Number for each state according to specific range of values for source/destination time to live.
32. **Ct_dst_ltm:** Number of connections of the same destination address on 100 connections according to the last time.
33. **Ct_src_dport_ltm:** Number of connections with the same source address and destination port in 100 connections according to the last time.
34. **Ct_dst_sport_ltm:** Number of connections with the same destination address and source port in 100 connections according to the last time.

35. **Ct_dst_src_ltm:** Number of connections with the same source address and destination address in 100 connections according to the last time.

36. **Is_FTP_login:** If the FTP session is accessed by the user via username and password value takes 1 else 0.

37. **Ct_FTP_cmd:** Number of flows that have FTP commands.

38. **Ct_flw_HTTP_mthd:** Number of flows which have HTTP related methods (e.g., Get, Post)

39. **Ct_src_ltm:**Number of connections with the same source address in 100 connections according to the last time.

40. **Ct_srv_dst:**Number of connections with the same service and destination address in 100 connections according to the last time.

41. **Is_sm_ips_ports:** If source equals to destination IP address and port numbers are equal, this variable takes 1 else 0.

As it has been mentioned before, each sample of the dataset is combined with a label which represents the status of the network, normal or under attack, and another extra label which determines the exact category the attack belongs to. The list of the documented attack types is:

- **Fuzzers:** Attempts of provoking a failure in a program by feeding it with random data.
- **Analysis:** Subset of attacks which include port scan, spam and HTML file penetrations.
- **Backdoors:** Attacks which try to bypass stealthily system security in order to access a computer o its data.
- **DoS:** Attack which tries to compromise the availability of a resource.
- **Exploits:** Attack which the attacker knows security flaws from which advantage can be taken of.
- **Generic:** Attacks against block cipher algorithms, without knowledge of the key size.

- **Reconnaissance:** Attacks used for information gathering.
- **Shellcode:** Attacks which try to integrate malicious code as the payload in order to exploit a software vulnerability.
- **Worms:** Attacks where the attacker tries to self-replicate in order to infect other computers.

3.2 Feature selection

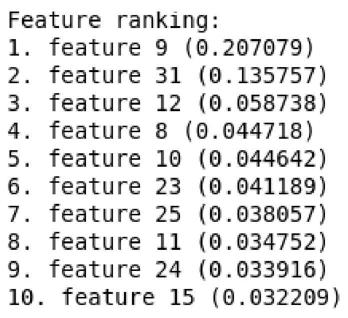
One of the goals of the feature gathering process is to make it as less time consuming as possible in order to perform real-time alike, although the main objective is still to develop a model which is as effective and accurate as possible. In order to address the first goal without compromising the other one, the approach that has been taken is the reduction of the number of features used selecting only the most important ones to perform the task at hand.

3.2.1 Feature importance with Sklearn

The first objective is to analyze the features and check their importance for the inner workings of the ML models. The *Sklearn* library implements many ML algorithms and, some of them can be used to determine the importance of the features. Feature importance can be defined as the score given to each of the input variables of the model based on the relevance they have on the prediction. In this case, two Sklearn functions will be used for this task, feature importance and permutation importance, both based on a Random Forest classifier.

The results are depicted in Figure 3.2 where the 10 most important features, together with the importance associated to them, are shown. The most important one of those listed in the enumeration 3.1 is the 8th feature, which relates to the STTL, source Time-To-Live. This feature got a 0.2 importance, which is a high value compared to the rest of the variables. Other variables with significant importance are the 31st and the 12th, which relate to the *ct_state_TTL* and the *Dload* features respectively.

The second approach to calculate the features importance is to use the *permutation method*. This method trains a base classifier with all the features, and others in which one feature



```
Feature ranking:
1. feature 9 (0.207079)
2. feature 31 (0.135757)
3. feature 12 (0.058738)
4. feature 8 (0.044718)
5. feature 10 (0.044642)
6. feature 23 (0.041189)
7. feature 25 (0.038057)
8. feature 11 (0.034752)
9. feature 24 (0.033916)
10. feature 15 (0.032209)
```

Figure 3.2: Feature importances for UNSW-NB15 dataset.

is removed. This permutation is done for all the variables, therefore, it can be a time consuming process if the feature list is big. The feature permutation mean difference and standard deviation are shown in Figures 3.3 and 3.4.

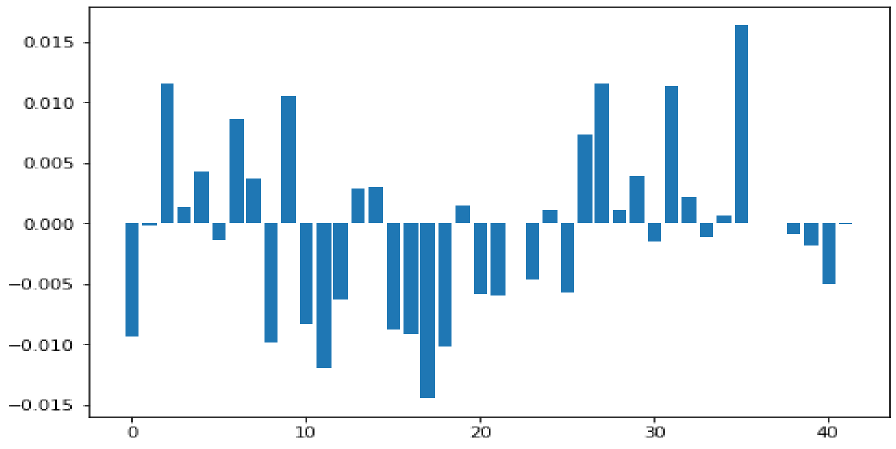


Figure 3.3: Feature permutation mean difference for UNSW-NB15 dataset.

As can be seen, the results achieved using the permutation method did not provide any relevant information, as the differences between the baseline and the permuted models are not big enough. In consequence, this method will not be used to perform the feature selection. In contrast, the feature importance approach, in a nutshell, has revealed some features which are relevant for the models decision process. However, these findings are not solid enough to create a new subset of features, as most of the features were of a similar importance.

3.2.2 Greedy algorithm

As the previous method did not provide good enough results, a new approach based on greedy algorithms will be taken. Greedy algorithms are commonly defined as simple,

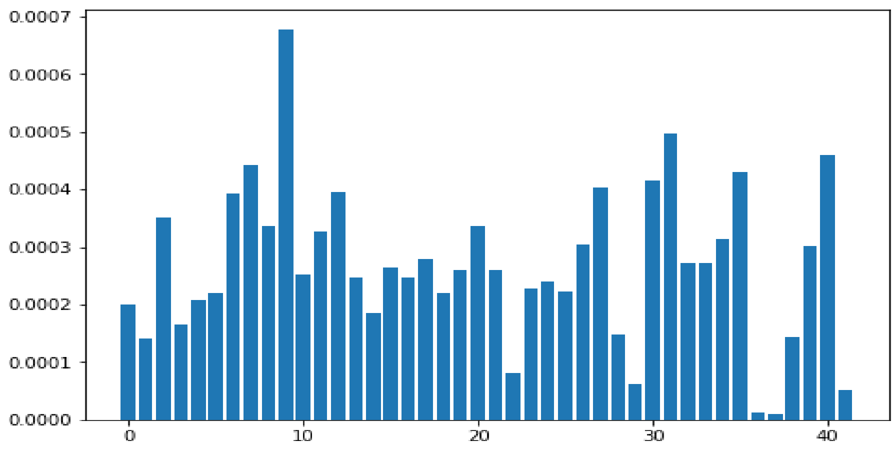


Figure 3.4: Feature permutation standard deviation for UNSW-NB15 dataset.

intuitive algorithms which find a solution for the task to be solved, until it is not able to be improved, in fact, this solution could be an optimal solution for the whole problem. Thus, this heuristic is used for optimization tasks where finding a solution by brute force is inefficient or when getting a sub-optimal solution is enough in order to solve the problem.

The common premise for the application of this algorithm is to have a set of elements to be selected for optimization. Those elements can be sorted using the value achieved in the final solution. Having the list of elements and the other restrictions of the problem set, the algorithm selects the element which can provide the highest increase of the value in the final solution. This step is iteratively repeated until the solution cannot be improved further with the choices made by the algorithm.

For the task of feature selection over the UNSW-NB15 dataset, the features cannot be sorted according to their value for the final solution, therefore, the approach for this case is to randomize the list of features to be selected, and pick iteratively a feature of the randomized list. Then, a binary classifier is trained and evaluated according to the metric *F1 score*, as it is an appropriate metric when the data is unbalanced. The quality of the predictions made by the classifier will be the criterion which will decide which features will be selected.

As can be seen in Figure 3.5, this sequence is repeated for all the elements on the list, and the list of features will be updated at every iteration with a new feature. At each iteration, it has to checked if F1 score achieved by the new classifiers improves the best F1 score recorded. In order to select the most relevant features overall, and not rare outcomes, the algorithm will be executed 30 times.

```
feature  f1_score
     32  0.000000
      1  0.550489
     38  0.560000
     10  0.560611
     16  0.564465
     13  0.567574
     18  0.768364
     33  0.825443
     27  0.825631
     34  0.839260
      6  0.876277
      3  0.884105
      7  0.888465
     12  0.890898
      4  0.894302
```

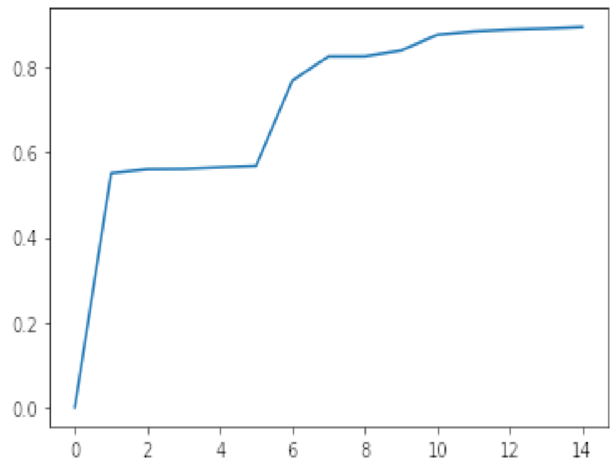
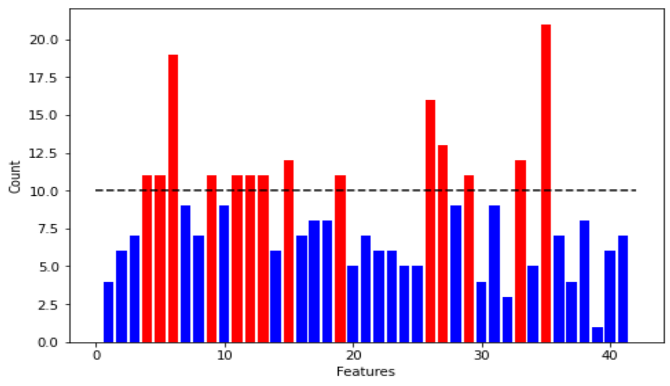


Figure 3.5: An example of a greedy algorithm execution with the visual representation of the performance increase associated with the increase of chosen features.



```
Selected features:
 [ 4.   5.   6.   9.  11.  12.  13.  15.  19.  26.  27.  29.  33.  35.]
```

Figure 3.6: The choices made by the greedy algorithm over 30 executions.

In Figure 3.6 the results of the execution are shown. For the final selection, the criterion is that the feature must have been selected at least 10 times by the algorithm, therefore, it must have appeared on $\frac{1}{3}$ of the solutions. The rationale behind this, otherwise arbitrary criterion, is that features which are relevant for the model will tend to appear more often, even if the list is randomized, and including them will improve the performance of the selected metric.

3.3 Selected features

Having the feature importance and greedy algorithms executed, there is enough information to create a ranking of the features which contribute more notably to increase the overall fitness of the model. Hence, using the combination of the results given by both methods, the list of chosen features is the following:

- Source packet count(4).
- Destination packet count (5).
- Source bytes (6).
- Source TTL (9).
- Destination TTL (10).
- Source load (11) .
- Destination Load (12).
- Source loss (13).
- Source inter-arrival packet time (15).
- Source TCP window advertisement (19).
- Mean flow packet size transmitted by source (26).
- Mean flow packet size transmitted by destination (27).
- The content size of the data transferred from a HTTP service (29).
- Number of connections with the same source address and destination port in 100 connections according to the last time (33).
- Number of connections with the same source address and destination address in 100 connections according to the last time (35).

Having the features which will be used, the implementation of the NAM can be started.

4. CHAPTER

The Network Analysis module

In this chapter the Network Analysis module is described. This component of the project is aimed to be the main tool for extracting real-time data and statistics from the network traffic. This is achieved by plugging this component into a network which needs to be monitored. This data extraction and generation process can be defined as a pipeline, which involves three main stages:

- **The packet sniffer:** The program which will capture the information from the network.
- **The feature generation:** The algorithms, methodology and data structures used to generate the features for the ML module.
- **Feature post-processing and communication:** Final adaptations of the features in order to fit and enable the communications with the ML model.

4.1 The packet sniffer

A packet sniffer or analyzer is a software or hardware component whose main purpose is to intercept or log the traffic which is “visible” from that segment of the network. Often, classic solutions involve installing these appliances in wired shared networks, which can be monitored easily with one device. However, as the networks increase in size and complexity, this paradigm begins to become obsolete. The wired networks are combined with wireless networks, the packets can be encapsulated to hide information and the topologies of the networks where these subnets coexist become more and more complex, to a point in which only one device or program could not be able to monitor properly all the subnets, as its scope is limited to an extent.

This increase in complexity can be addressed following two approaches: using high-end appliances which have the required capacity to process the traffic of the whole network, or using a distributed platform which could involve many affordable components strategically positioned in the network with a centralized control module. The sniffer which will be explained in this chapter is aimed to be a low resource consuming program which can be installed in any machine of the network which has one or many network interfaces. The module is written using the C language and the *libpcap* library.

4.1.1 Libpcap

Libpcap[1] is a packet capture library which provides a high-level API for packet capturing implementations. The main advantage of this particular API is that all packets in the network can be intercepted, even the ones that are sent to other hosts. In fact, this library is the main component of the command-line tool *tcpdump*.

The workflow for most libpcap applications is the following:

- **Initialization:** The library components must be initialized providing an interface name or identification in order to attach it to the packet capturing program; if no interface is provided or the name is unknown, functions such as $pcap\_lookupdev()$ or $pcap\_findalldevs()$ are used for that purpose.

- **Open a capture handler:** Having the device specified, further processing can be achieved. The next step in the workflow is to open a capture handler, which can be attached to the recently obtained device or can be attached to a “dead” interface in order to dump the captured information into a file. In this case, the function $pcap\_open\_live()$ will be used, in order to generate a “packet descriptor”, which is the abstraction made by libpcap for the handler.

- **Filtering and compiling:** The filters in libpcap are specified as text string and can contain elements such as hosts, network addresses, ports, packet lengths; they even can represent more complex concatenations which define destination subnets, port

[1] https://www.tcpdump.org/

ranges or protocols. In this case, the goal is to intercept and monitor all the reachable packets and, therefore, no filtering expression will be used. Having the filter ready, the filter must be compiled using the *pcap_compile*() function, in order to generate the filter program the handler will use when receiving packets.

- **Link-layer type selection.** There are many technologies implemented for the link layer in which the header lengths differ from each other. In order to ease this process, the function *pcap_datalink*() is used, which returns the link type of the device the handler is attached to.

- **Setting the packet processing loop.** The main workload of the program is done by the packet capture loop. In order to setup the process, the functions *pcap_dispatch*() or *pcap_loop*() must be used, which allow the one-by-one picking of the packets (*pcap_next*(), *pcap_next_ex*()), or setting a loop to receive a batch of packets respectively. In this case, the loop method will be used together with a callback function to add the logic and the processing needed to generate the data samples.

- **Statistics:** As a part of the packet capturing process, the handler makes several calculations, such as received packet count, lost packets or lost packets due to interface failure. Getting these statistics is easy, as invoking *pcap_stats*() returns the current count of all those metrics.

4.2 Data Structures

The sniffer is the module which provides the functionality of extracting raw information from the network traffic. However, this information *per se* cannot be used to achieve the objectives mentioned in the specification of the project, and further processing must be done to extract the required features. This will be achieved by applying different techniques and algorithms to the captured data. Since not every feature is extracted the same way, the approach taken for each one is different, although the process can be categorized in three main subsets:

- **General features:** These features can be extracted from the headers of the packets without much, if any processing, and are collected continuously, as soon as the packets are analyzed

- **Processed features:** This subset contains the features which are generated by some calculation based on the general features or that need to obtain the information from the payload of the packets.
- **Post-processed features:** These features are generated based on calculations over some general features that change over time. To do so, a record of the last connections has to be maintained, and the value of the feature is computed just before the data is sent to the ML module.

Once the categorization has been made, the data structures which are going to contain the information to extract the features can be explained in detail.

4.2.1 Flows

The first concept that must be addressed in regard to the sniffing process is the concept of *flow*. In this case, a flow is defined as a bidirectional sequence of packets between two different IP addresses, and it is bound to the ports and transport protocol used for the exchange of information. This concept is used to separate and disambiguate its meaning from the commonly used term *connection*, as this could be interpreted in a strict context of TCP connection or could be misunderstood with the physical connection itself, which are not accurate for this purpose.

Therefore, the flow is implemented as a struct[2] which contains six elements, two char arrays of size 256 for the source and destination IP addresses, two integers for source and destination ports, one char pointer for the protocol used in the flow, and a struct which contains the information of the captured data, called sample data, which will be explained in the Section 4.2.3.

4.2.2 The circular buffer

Having the structure which defines the parameters used to describe each flow of information that traverses the network, the next objective is to define the data structure which will hold that information for further processing. In this case, the sniffer itself will act as a generator of information and the algorithms used for the feature extraction will act as consumers of that information. As this process can be seen as a consumer-producer model,

[2]https://en.wikipedia.org/wiki/Struct_(C_programming_language)

one of the most efficient approaches to communicate them is the use of a circular buffer where the most recent information will be available. This way, the process to store the raw data is separated from the process that analyzes it which is a very convenient feature regarding the module must be as time efficient as possible.

The struct itself is simple: three integers which contain the head, the tail and size of the buffer, and an array of flows. For this project, the aim is to replicate the data generation process used in the UNSW-NB15 dataset as accurately as possible. Therefore, as the number of connections taken into consideration for the post-processed features was 100, that is the number of flows which can be monitored at the same time in the buffer. However, the size of the buffer could be increased at will if more flows must be maintained or monitored simultaneously.

4.2.3 Sample data

This is the data structure which contains the biggest amount of information, as it is where all the headers and other additional information extracted in real-time is maintained. It is part of the flow struct, as it holds the captured information of each flow being monitored. The struct is defined to contain the following elements:

- Source and destination packet count (integer).
- Source and destination byte count (integer).
- Source and destination TTL (integer).
- Source and destination load (float).
- Number of packets lost in source–destination traffic (integer).
- Source TCP window advertisement (integer).
- Source packet inter-arrival time (double).
- Total packet inter-arrival time (double).
- First timestamp, current source timestamp and current timestamp (struct timespec).
- Source and destination mean packet size (float).
- Same source and destination IP addresses count per 100 connections (integer).

- Same source IP address and destination port per 100 connections (integer).

The struct timespec is a data structure defined in *sys/time.h* which is used to get the time after calling functions such as *nanosleep*() or *clock_gettime*(). It is formed by a field of type *time_t* that contains the seconds and a field of type *long* that contains the nanoseconds.

4.2.4 Directional info

Being the flow a bidirectional abstraction for the exchange of packets, there is some information that cannot be categorized and updated without some processing. This issue becomes more noticeable when addressing information such as the TTL or the byte count, because those are available in the packet header but it is required the directional information (IP addresses and ports) to be updated properly.

For that reason, the *directional info* structure is used, being its purpose to contain the data which needs that directional information in order to be updated properly within the flow. The struct is formed by three integers which represent the byte count, the TTL of the packet and the packet loss.

4.2.5 Samples

This data structure contains the information which is going to be sent to the ML module and, therefore, its format has to be the same as the one defined in the Section 3.3. This means that this data structure is only going to be used as the last step of the pipeline of processes which conform the NAM, just before the communication phase.

4.3 Feature extraction

In order to send samples to the ML module, these must first be created within the NAM, feature by feature. As mentioned above, these features are those explained in Section 3.3. In this section, the procedure used for the extraction and generation of each feature will be explained in detail.

4.3.1 General features

As mentioned before, features within this subset can be easily found in the packet headers or can be found with methods which are not complex nor time-consuming. The features and the procedure used for the data extraction are explained below:

- **Source and destination packet count:** These are counted each time the capture loop handler function is called, as the function is called on every packet processed. This feature is an example of directional info, however, as the value associated is increased by one each time it is addressed, this information can be updated at the end of the handler function, having the IP addresses and ports available.
- **Source total bytes:** This information is easily extracted from the IP header *ip_len* field. This is directional information, therefore, the handler must control the direction of the flow for this feature too.
- **Source and destination TTL:** This information is also directional and it is extracted from the IP header field *ip_ttl*.
- **Source packet loss:** The packet loss is calculated using $pcap\_stats()$ on every iteration of the capture loop in order to get the received packet and the lost packet counts, while having a record on the packet statistics in the previous iteration. The difference between the two lost packet values will be saved on the feature. Having the value, the usual control must be done to update it at the proper direction.
- **Source TCP window advertisement:** The value related with this feature is found deeper in the packet headers. In this case, it can be found only on the TCP headers, in the field named *window*. As this information is also directional, it only needs to be saved in the source $\rightarrow$ destination direction.

4.3.2 Processed features

The features that belong to this subset are the ones which need some processing using the general features or those that are more complex to extract. The features are the following:

- **Source and destination load:** Theses features are calculated as the division of the total byte count transmitted from the source/destination and the duration of the flow.

This metric represents the total bytes transmitted per unit of time (B/s). The total time is calculated using the timestamps mentioned in the Section 4.2.3.

- **Source inter-arrival packet time:** In order to calculate the value of this feature the difference between the arrival times of the previous and the current packet is computed. This is stored for each packet when it is transmitted, and when the flow has to be sent, the mean inter-arrival time is calculated using the number of packets sent by the source. This feature has to be represented as milliseconds.

- **Source mean packet size:** This feature is calculated by dividing the total byte size transmitted from the source and the source packet count.

- **Destination mean packet size:** This feature is calculated by dividing the total byte size transmitted from the destination and the destination packet count.

- **HTTP response size:** This feature is extracted from the payload of a TCP connection, which increases the time required to be processed. Moreover, the payload must be inspected as the only information which is really needed is the *Content-Length* field, which contains the byte count within the HTTP content. This information must be only accessed under an oddly specific condition: when a TCP connection is used and the destination port is 80. The initial intention was to apply the same procedure for the HTTPS protocol (port 443), however, this is not possible, as the communications are encrypted, leaving the TCP payload unreachable to extract the header field needed for this feature.

4.3.3 Post-processed features

Until this point, all the features can be extracted at any time during execution. The main distinction between the features of this subset and the previously described ones is that these post-processed features must be calculated just before the generated data sample is sent to the ML module. These two features are calculated based on the state of the buffer at the time the sample is deployed and the values that can contain are within the range [0,100]. These features are the following:

- **Same source and destination IP addresses count per 100 connections:** This feature is calculated comparing one flow contained in the circular buffer with the rest of the flows. On each comparison, the algorithm checks if the IP addresses involved in the flow, both source and destination, are the same in both flows. If that condition is met, the value of this feature is increased by one.

- **Same source IP address and destination port per 100 connections:** This feature is similar to the previous one as the IP addresses are checked, but only in source, and in this case the destination and source ports are also checked. As before, if the condition is met, the value of this feature is increased by one.

As can be seen, the moment when these features must be calculated must be the instant before the sample is sent, not only because is the only way to calculate the post-processed features properly, but also because this is the most efficient way to calculate the post-processed features.

4.4 The handler program

The last element of the pipeline of processes which conform the NAM is the custom handler function. This function will be executed on every loop of the packet handler, therefore, this function needs to implement all the control algorithm and the general feature extraction from the data:

4.4.1 General feature extraction

While the control algorithm is working, the information from the packet is being extracted for each iteration. In order to achieve this, the loop uses 3 variables which act as placeholders for the captured information: *thisFlow* contains the general features and the IP addresses, ports and protocol, *extra_info* contains the directional info of the packet, and *timestamp* contains the information saved within the timespec struct.

The gathering of information during the loop is achieved mainly by inspecting the headers of the packet, and the information which is extracted varies depending on two main factors: the transport protocol used and the ports which are targeted on the communication. As can be seen in the Figure 4.1, the feature extraction occurs in the first half of the algorithm, where the yellow coloured boxes represent updates on the flow data structure and the blue coloured one relates to the data structure which contains the directional information.

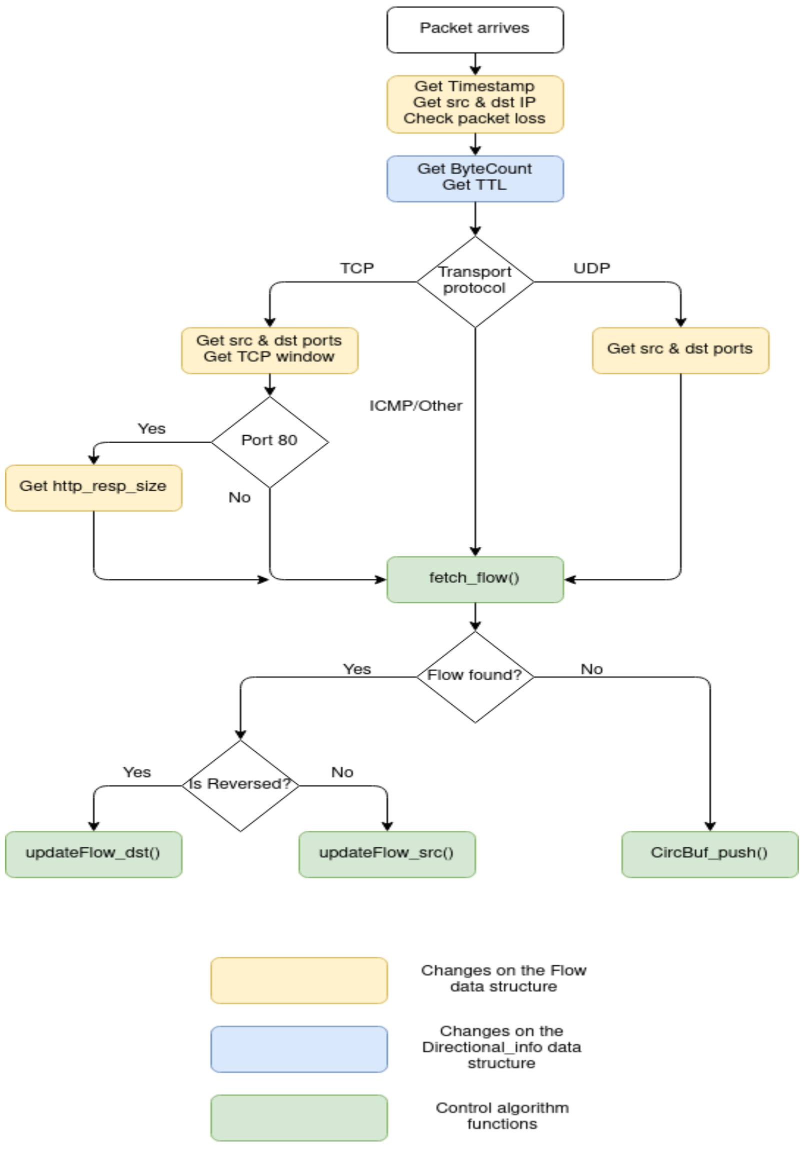


Figure 4.1: Flowchart of the handler function.

4.4.2 Control algorithm

On every loop the handler function needs to identify the IP addresses and ports involved in the packet transmission. Having these and the transport protocol associated to them, the control algorithm can decide whether the flow has been previously added to the circular buffer or not. In case the flow has been already been registered, the algorithm checks the direction of the packet in the flow, in order to update properly the directional information in the sample data struct within the flow.

One assumption made on this algorithm is that the packet which is checked on every iteration is the first of the flow associated with it, therefore, if the algorithm checks that it is in fact the first appearance of the flow, the assumption will be correct and the flows source–destination direction will be the one represented within the information of that packet. If the flow related to the packet has already been checked, the direction of the packet will be evaluated by comparing its directional information with the flows already stored in the buffer.

In order to implement this algorithm, the following functions have been programmed:

- ***CircBuf_push():*** This function is called in order to push a new flow into the circular buffer. Its input parameters are the flow to be pushed, the directional information and the timestamp of that iteration. Firstly, some variables of the flow are initialized, and the directional information and timestamp are added to the flow. Having the flow initialized and filled with all the necessary information, the circular buffer can be updated with the new flow in the position the tail is on. After updating it, the tail is incremented in order to point to the next position.

- ***CircBuf_pop()*****:** This function returns the flow the head is pointing to, and updates the head to the next position.

- ***fetch_flow()*****:** This function receives as parameter a flow generated from the received packet information. The circular buffer is iterated, and each flow is compared to the new flow, in order to check if it belongs to that flow or not. If a match is found, the function returns the index of the fetched flow. If no match is found, an error code is returned.

- ***isReversed()*****:** This function receives as input two flows, being one of them the flow returned by the previously explained function and the other one the flow associated with the arrived packet. Therefore, if this function is called, it is certain that both flows are related, hence, the only check which needs to be done is the direction the new packet belongs to. This is achieved bay comparing both flows IP addresses, ports and protocol. If all the information is equal, it means the packet will belong to the source–destination direction. In case the source IP address and port of the flow are equal to the destination IP and port of the packet and the destination IP address and port of the flow are equal to the source IP and port of the packet, the packet will belong to the destination–source direction.

- ***updateFlow_src()*** & ***updateFlow_dst()*****:** These functions are called once the *isReversed()* function has been called. As at this point, he direction of the packet is known, these functions can update the variables of the flow accordingly. Both functions have the same input parameters: the flow that has been generated from the arrived packet, the directional information of the packet, the index which points to the position the flow is saved in the buffer and the timestamp of that packet. The update consists of merging the new information with the information already saved, increasing the packet counts, the total byte quantities, timestamp calculations, and others.

In the Flowchart 4.1 the control algorithm is represented within the second half, once the information has been extracted from the packet. The functions interact in the way described in the figure in order to implement the logic of the circular buffer updates and the updates of the existing flows with the recently extracted new information. The colour associated to these funtions is green.

4.5 Sampling the data

As the number of iterations of the loop increases, the circular buffer will be filled with the information of the flows, and when the event which requests a diagnostic of the network status occurs, this module needs to transform those flows in a data structure the ML module understands.

Firstly, the processing starts by extracting the flow from the buffer. This can be achieved by simply calling the *circBuf_pop()* function, which will free the position the flow was occupying. Secondly, even if the flow is filled with data, it is not complete yet, as the only features it contains are the general ones. Hence, the processed and post-processed features need to be calculated in order to complete the information. This is achieved using a function called *Calculate_Features()*, which receives as input a flow and returns the same flow filled with the information required to complete the sample. The process performed on the flow is the following:

- **Post-processed features:** The flow is compared with the rest of the flows in the buffer, and the same_src_and_dst_ip and same_src_ip_and_dst_pt are generated according to the comparisons made.
- **Time calculation:** The total communication time is generated by calculating the difference between the first and the last timestamps. This total time will be used to calculate some time-dependant features.

- **Source inter-arrival time:** Its value is the division of the total arrival time of the packets arrived to source and the total source packet count.
- **Source and destination mean:** It is calculated dividing the total byte count and the total packet count for both source and destination.
- **Source and destination load:** Calculated by dividing the total byte count with the total communication time, for both source and destination.

Lastly, having the flow complete, the sample itself is generated by saving only the features the model needs in a data structure prepared specifically to be sent to the model.

4.6 The multithreading paradigm

Once all the functionalities have been developed, the program seems to work fine, in the strict sense of the term, as the program fulfilled its purpose, even though it was not performing real-time alike. In order to solve the mentioned issue, the approach taken was to use an even more fine-grained design, dividing the whole program into smaller processes (threads). This way, the program will also take advance of the multi-core paradigm present on modern processors and increase the performance accordingly. The pthread [3] library is the one chosen to implement the processes in different processing threads and to synchronize them and avoid any kind of race condition or deadlock.

Therefore, the structure of the module has to change to support this new paradigm. The objective remains the same, although the interactions between modules are different. The packet sniffing process, the handler described in the Flowchart 4.1 and the sampling process are executed in different threads, as it is shown in the Figure 4.2.

4.6.1 Structural changes

In order to support the new paradigm, all the processes which form the NAM will be created using the pthread library. Now that each part of the process is implemented as an independent thread, and as the diagram 4.2 suggests, there is no need to strictly use one thread per task, as there is the possibility of generating this processes on demand. This approach allows a much higher degree of flexibility and increases the versatility of the

[3] https://man7.org/linux/man-pages/man7/pthreads.7.html

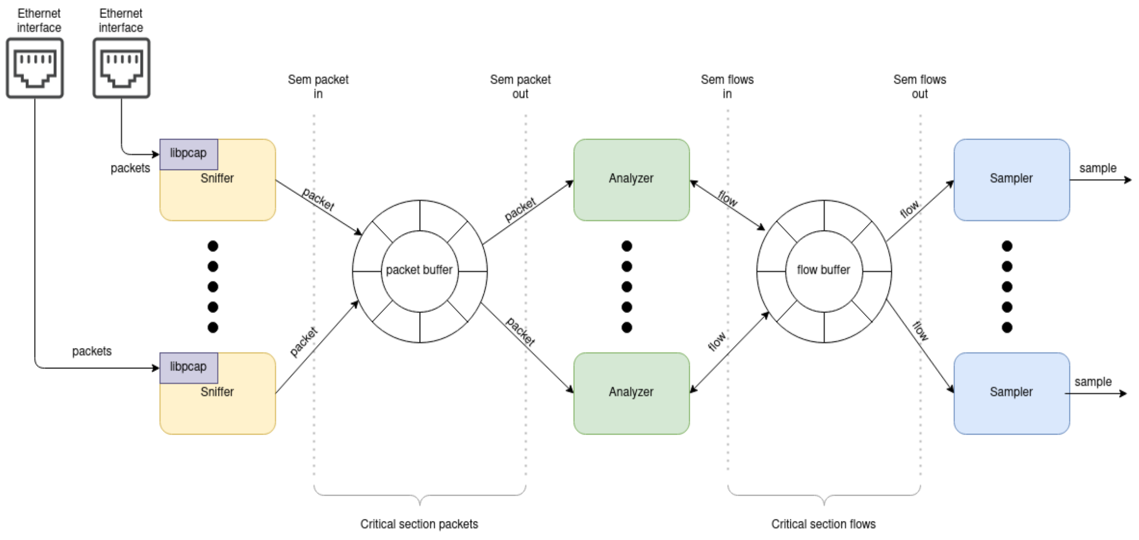


Figure 4.2: The structure of the multithread version of the Network Analysis module.

module, as it can be customized and tweaked in order to adapt to the necessities of the user and network.

The first concern about the division of the original program into smaller processes are the data dependencies, because when all the data structures were initialized and updated, there was no need to add any layer of abstraction. However, now that the programs are held within different threads, it is required to rethink the way shared structures are accessed.

The main change in the data structures is the addition of a new circular buffer, whose purpose is to communicate the capturing and the analyzing processes. This structure will be used in a consumer-producer fashion, where the sniffer will feed the buffer with packets and the analyzer will consume them. Summarizing, the main change has been to organize how the main parts of the modules which are now threads will communicate and synchronize.

4.6.2 Synchronization

Having the threads working, the resources are being accessed, but not correctly, as there are not mechanisms which control the access to the shared resources, leading to race conditions and deadlocks on the program.

The common approach for synchronization on UNIX-like systems is using mutexes from the pthread library, which provides an easy way to implement mutual exclusion mechanisms in order to protect critical sections within the code. This is a good approach for the access of the buffers, however, the issue of the circular buffers in this case is their capacity, as the aim is to process all the packets and flows without overwriting any of them. Moreover, on top of the mutexes a dual semaphore setup will also be used for both of the buffers, as the semaphore provides a count of accesses that can be done until the section is blocked, and the thread waiting will gain access to the section when other thread increases the count.

Having that system allows a correct and synchronized implementation of the consumer/producer paradigm, as one of the semaphores will control the data injection in the data structure, and the other semaphore will control the data extraction from the data structure. As can be seen, both semaphores must be updated when pushing or popping data in order to manage correctly the access to the resource. The initialization of the "in" semaphore corresponds to the maximum number of elements the buffer can hold, and the "out" semaphore is initialized to 0. This initialization is done this way in order to allow the producer to generate the data which will inject into the buffer directly as long as the buffer has space, and the consumer will have to wait for the data to be injected in order to access the buffer. Therefore, the pushing of the data will decrease the "in" semaphore count and increase the "out" counter by 1, and the popping of the data will decrease the "out" counter and increase the "in" counter. This approach can be applied for both packet and flow buffers.

Once the synchronization is being managed correctly, the whole system works perfectly and takes advantage of the multithread paradigm. Hence, the module is now able to start many threads of each of the parts of the system, allowing, for example, the increase of the performance of the packet analyzing process by taking advantage of a multi-core system or, even monitoring multiple network interfaces (subnets) at the same time. The same can be done with the sampler, being now possible to extract the features of the flows concurrently in many threads and use one or many ML modules to perform the classification of the samples. In general, we can conclude that the use of a multithreading paradigm has increased, not only the theoretical peak performance of the system, but also the scalability.

5. CHAPTER

The Machine Learning module

This module of the project will make the decisions according to the status of the network. This task can be simplified to a function which is fed with a sample (x), and gives a prediction (y). The function (F) which connects the input and the output is unknown, and is the task which the machine learning algorithm will address by training it with the data.

This anomaly detection [Denning, 1987] approach based on machine learning has several advantages [Shon and Moon, 2007] [Bhuyan et al., 2014] compared to a traditional, signature based, IDS implementation:

- Attacks which could not be detected because their pattern is not registered can be detected with a correct modeling of the normal behaviour of the network.
- It offers flexibility and options of customization in order to adapt the model to a specific network.
- The hidden representation of the data could reveal indicators of which parameters of the network can be optimized in order to defend better against some attacks

However, it is also known that anomaly detection based IDS are prone to give significant counts of false positives [Clarke et al., 2008] [Mell et al., 2001], as the modelled "normal" behaviour could not fit every usage of the network In fact, this is forced to minimize the quantity of false negatives, as these would represent all intrusions which were not detected as such.

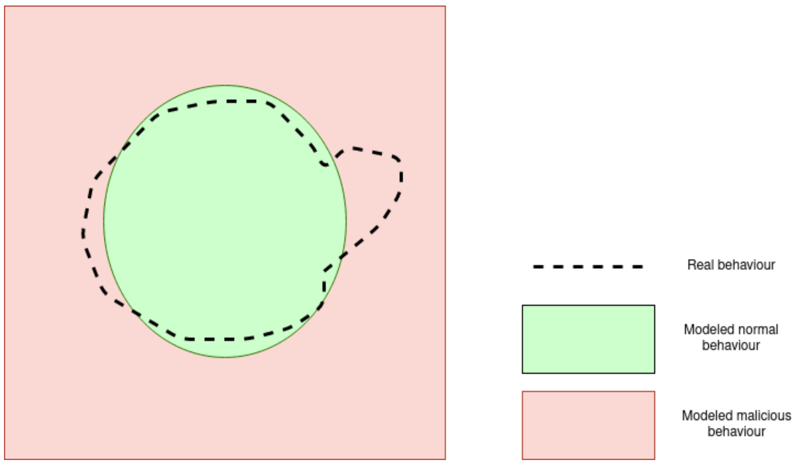


Figure 5.1: A graphical representation of an anomaly detection based IDS system.

As shown in Figure 5.1, the modeled "normal" behaviour is not perfect in the sense it does not adapt perfectly to the real behaviour of the system, in this case it is clear that plenty of situations will alert the system as an attack, although as mentioned before, this is more desirable than passing malicious behaviour as normal traffic.

Theoretically, if the models capacity is big enough to hold all the information, feeding it with enormous quantities of data, will allow it to fit more accurately the real distribution of the data, in this case, the one considered as the real behaviour of the system.

5.1 The metrics

Considering this anomaly detection approach and having explained the importance of choosing appropriate metrics, the metric used for the evaluation of the binary classification task, which involves discriminating malicious behaviour from normal behaviour, is F1 score.

F1 Score is a measure for accuracy, defined as the weighted harmonic mean of the precision and recall. Precision is defined as the proportion of positive results which are in fact positive, and recall measures which choices between all the positive ones are relevant. When it is said it is the harmonic mean, this emphasizes the idea of focusing the calculation of the mean in order to discriminate large outliers and increase the importance of small ones, which in practice translates to a more balanced calculation compared to the standard arithmetic mean.

Therefore, the F1 score can be written as:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Where

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

For the task of multiclass classification the model can return more options up to 10 different categories, leaving aside the previously mentioned paradigm, where only two options were available and knowing the nature of the binary predictions was rather important, for the multiclass case the setting is changed, as the exact category must be selected. The metric used for measuring the performance on the multiclass task will be also F1 score, as the dataset is unbalanced, and this form of balanced accuracy can extract more information from the results.

5.2 The algorithms

On this algorithm trial phase, 3 different approaches will be taken to solve the task. The algorithms which will be used are a Multi Layer Perceptron (MLP), a Random Forest (RF) and a K Nearest Neighbor approach (KNN).

5.2.1 MLP

The Multi Layer Perceptron is a neural network which is conformed by perceptrons stacked in layers that can be connected with each other, which it is combined with a function to propagate the errors in order to force the model to learn. This whole paradigm is the common approach for all the Deep Learning approaches [Goodfellow et al., 2016].

On the one hand, this increase of capacity is beneficial, and it makes the model really versatile for many tasks, and given sufficient amounts of data, the model can be an accurate representation of the real data.

On the other hand, this increase of capacity also implies an increase of complexity, when using the perceptron, the function was modified by adjusting the boundary to fit better the data. However, as perceptrons are layered and connected with each other, now the errors must be backpropagated in order to update the weights of all neurons, using loss functions and optimizers. This complexity translates to a increase on the time and other resources to train the model.

As a concept, backpropagation [Rumelhart and Williams, 1986] is an algorithm which calculates the gradient of the loss function in order to update the weights of a feedforward neural network. This calculation is done iteratively, and is chained from the last layer to the first layer. This algorithm is widely used for supervised learning tasks.

The loss function is a concept extracted from optimization, where a number which is representative of the outcome of a problem that needs to be optimized is calculated, and the main task is to reduce its cost, as a general idea. In machine learning, this calculation is related to the errors committed by the algorithm when fitting the data, which must be minimized in order to improve the predictions of the model.

The optimizer is an extra element which combines with the loss function. Its main purpose is to update the weights according to the optimizer algorithm when the loss function is calculated. Therefore, it can be said that the loss function is the mathematical indicator of the models performance fitting the data, and the optimizer is the algorithm which performs the actual changes in the weights. Each optimizer has its own characteristics, and these strongly determine their suitability for the kind of optimization to be done, so often the usage of one or another can drastically change the performance of the model.

The actual implementation of the MLP will be done using Keras[1] library, as it provides a condensed, practical and convenient API which will ease the implementation of the model, without renouncing to the powerful Tensorflow[2] backend, as its integrated within the library.

5.2.2 RF

As mentioned before, this forest algorithm is formed by decision trees [Rokach and Maimon, 2008], which are a machine learning predictive model which given a set of observations, in this case, a vector of features, is able to decide the target value associated to that set of observations. Depending on these target values the decision tree is used for classification or regression, discrete values will relate to classification trees and real values will be used for regression. The main advantage of this algorithm is its readability, as the output of the algorithm can be easily read given the sample and the inner decisions made on the

[1]https://keras.io/
[2]https://www.tensorflow.org/

branches, as these decision-making process resembles to the one humans use. However, decision trees often suffer from overfitting, as the learning process derived from the algorithm often makes the trained tree excessively adapted to the training data, which works in detriment of the generalization of the model.

Besides, Random Forest is part of a subset of algorithms related to the Ensemble Learning [Rokach, 2010]. This approach benefits from the use of a high quantity of instances of learning algorithms to obtain better performance overall than each of the algorithms alone. For the case of the Random Forest, the decision trees generated are considered equally for the last solution boostrap aggregating, which essentially consists in weighting equally all trees. In addition to this weighting concept, the samples given to the models using this type of ensemble are randomized and replacement is allowed between samples, which contributes in the gain of overall robustness in the model, however, its main benefit is the reduction of overfitting and variance.

One of the properties of this algorithm, in fact, is that it can straightforwardly be used for a feature importance task, as the bootstrap training techniques combined with decision trees give an easily readable solution. This is achieved by fitting an standard Random Forest classifier, and then some other classifiers are trained with perturbed versions of the dataset, and the error is calculated and compared to the baseline error. This is applied over all the trees that conform the forest, and the results are averaged and the standard deviation is calculated. In fact, this technique has been used in section 3.2.1 for the analysis of the UNSW-NB15 dataset.

In practice, this algorithm can be easily implemented using Sklearn [3] library, which is going to be the one to be used in the experimental analysis of the algorithms.

5.2.3 KNN

The main singularity of this algorithm is that it is non-parametric, as the model does not use parameters, it uses the feature space instead, which consists on the n-dimensional representation of the data, where n is often the number of features each data sample has. Representing the samples as points, their distance is commonly calculated using Euclidean distance. Given the nature of this calculations, if the dataset is normalized the performance of the algorithm can be increased significantly.

The way this algorithm is executed shows its main flaw. The whole dataset must be rep-

[3]https://scikit-learn.org/

resented as points, and the distances must be calculated with each new query made, this process can get significantly slow as feature count increases, so does with extremely large datasets. Besides, if the K parameter is not selected properly the algorithm will not be effective enough, given as a general rule that small K values tend to simplify the calculation and decision process reducing the time needed to execute the algorithm, although accuracy can be reduced due to the lack of close data points which are not being considered.

On the other side of the spectrum, high K values will stabilize the solution as the averaged criterion extracted from a lot of points often relates to better decision process, even though excessively high values of K will include in the decision process points with low significance for the decision, which will cause a detriment in the performance of the algorithm, not to mention that the bigger the K value, the longer the algorithm will take to find a solution. Therefore, the best option is to choose an in between approach, in order to benefit from both good averaged decisions and reasonable computing time.

Another relevant point to discuss is that in datasets such as this UNSW-NB15 dataset, with highly unbalanced representation of the labels, the algorithm is prone to give more weight for the decision to the most frequent labels. This can be overcome with a weighted consideration of the different labels of the dataset.

Nevertheless, the algorithm is also known to be highly versatile, as can be adapted to a wide variety of tasks, and its simplicity is also one of its key strengths, and the fact that is non-parametric is another feature to be considered, as this removes the need to tune parameters.

This algorithm is also implemented in Sklearn library, and is the one which will be used in the experimentation.

5.3 Experimentation

Before choosing the algorithm which will be used in the module, the algorithms will be trained and tested with the UNSW-NB15 dataset and their performance on the task of binary and multiclass classification will be evaluated. The metric to evaluate the quality of the models will be F1 score, as the dataset is unbalanced, this approach should give more relevant information about the predictions.

5.3.1 Experimental framework

The main criterion which will decide the algorithm to be selected is the greatest F1 score, although another factors such as training time can be addressed in case a deeper analysis is needed. The optimal configurations and parameter tunings will be found using a trial and error method, therefore, only the configurations that give the best results for each algorithm will be reported in this document.

5.3.2 Setup

For the MLP, the model has been implemented using Keras, and the configuration which has given the best results has been a 3 dense layer architecture with 1000, 500 and 1000 neurons on each layer for the binary classifier and a 3 layer approach with 800, 400 and 100 neurons for the multiclass approach. A dropout layer has also been added on the last layer in order to improve generalization, with a dropout rate of 0.3 .The last layer of the binary classifier is formed by a single neuron, which will represent the prediction. The last layer of the multiclass classifier is formed by a 10 neuron layer. A sigmoid function has been chosen as the activation function for binary classification on the last layer, and a softmax approach has been taken for the multiclass one. The chosen loss functions are binary cross-entropy and categorical cross-entropy, respectively. The optimizer that has been used is Adam, for both classifiers, and the learning rate is 0.5 for the binary classifier and 0.1 for the multiclass classifier. In order to train the model, the training has been repeated for 20 epochs for both classifiers, and the chosen batch size is a 400 sample one.

The Random Forest has been implemented using the Sklearn library, with the ensemble module specifically, and the number of estimators used has been 500. The used KNN algorithm is also a Sklearn implementation taken from the neighbors module, with a K value of 5.

5.3.3 Results

With the setup proposed in subsection 5.3, the algorithms were trained and tested with the already partitioned dataset, the results are shown in Figures 5.2 and 5.3

As a general rule, it can be said that the algorithms perform reasonably, with a clear superiority of the Random Forest algorithm, which has a near perfect F1 score on training

```
[0.8539816737174988, 0.9981692815713381, 0.935742353471236]
[0.7051085829734802, 0.8889596750913307, 0.8218001877640432]
```

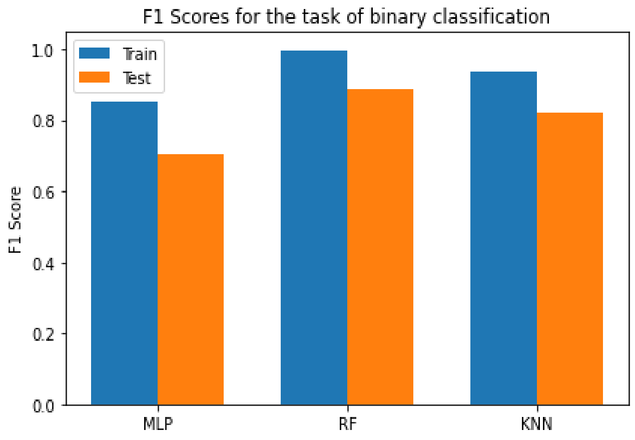


Figure 5.2: F1 score comparison among MLP, RF and KNN on the task of binary classification.

and the highest F1 score in the test. Therefore, it can be assumed Random Forest is the best algorithm for the binary classification task.

```
[0.550411018801924, 0.9075344614208884, 0.7448001323136061]
[0.4298328717873974, 0.5857382305786328, 0.5614341932662877]
```

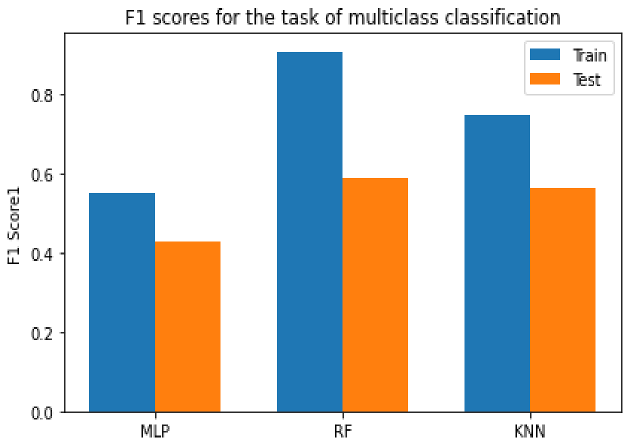
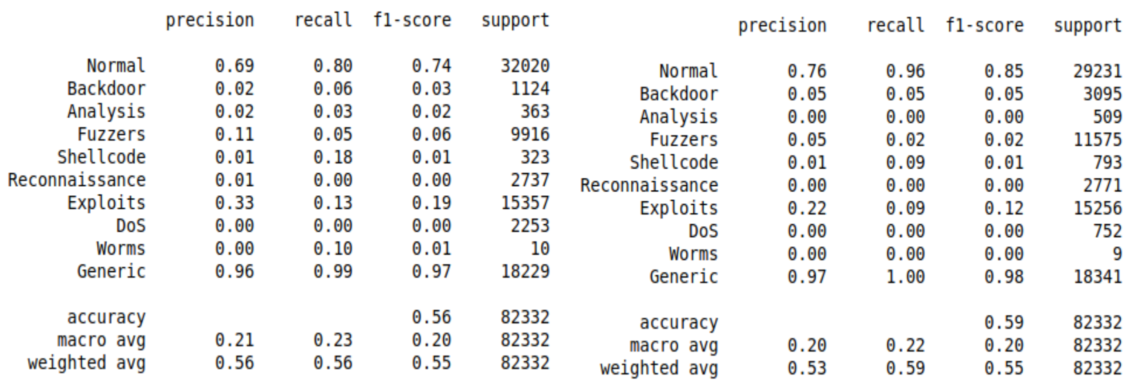


Figure 5.3: F1 score comparison among MLP, RF and KNN on the task of multiclass classification.

However, for the task of multiclass classification, the initial results do not show a clear superiority between RF and KNN, as both perform similarly on the test set, even though RF has generalized worse than KNN, as the difference between their training F1 scores is greater, so deeper analysis must be carried in order to reach a verdict.

As shown in Figure 5.4, even though both algorithms produce a similar F1 score, the distribution of the decisions made by both algorithms is significantly different. For both



```
                precision    recall  f1-score   support

        Normal       0.69      0.80      0.74     32020
      Backdoor       0.02      0.06      0.03      1124
      Analysis       0.02      0.03      0.02       363
       Fuzzers       0.11      0.05      0.06      9916
     Shellcode       0.01      0.18      0.01       323
Reconnaissance       0.01      0.00      0.00      2737
      Exploits       0.33      0.13      0.19     15357
           DoS       0.00      0.00      0.00      2253
         Worms       0.00      0.10      0.01        10
       Generic       0.96      0.99      0.97     18229

      accuracy                           0.56     82332
     macro avg       0.21      0.23      0.20     82332
  weighted avg       0.56      0.56      0.55     82332
```

```
                precision    recall  f1-score   support

        Normal       0.76      0.96      0.85     29231
      Backdoor       0.05      0.05      0.05      3095
      Analysis       0.00      0.00      0.00       509
       Fuzzers       0.05      0.02      0.02     11575
     Shellcode       0.01      0.09      0.01       793
Reconnaissance       0.00      0.00      0.00      2771
      Exploits       0.22      0.09      0.12     15256
           DoS       0.00      0.00      0.00       752
         Worms       0.00      0.00      0.00         9
       Generic       0.97      1.00      0.98     18341

      accuracy                           0.59     82332
     macro avg       0.20      0.22      0.20     82332
  weighted avg       0.53      0.59      0.55     82332
```

Figure 5.4: Side-to-side comparison of KNN (Left) and RF (Right) on the multiclass classification.

algorithms, the generic attack detection has been almost perfect, however, the RF algorithm has performed better recognizing normal traffic, as both its precision and recall have been better compared to KNN ones, which means RF has performed better discriminating malicious traffic overall, as it has scored better on the precision metric, in fact, this is an important point in favor of RF, as it allows a lower false negative rate. As a good feature of KNN, must be mentioned that has performed better recognizing analysis, fuzzer, exploit, reconnaisance, DoS and worms based attacks, although this difference is not significant enough to tip the scales in favor of KNN. Therefore, RF is a better suited algorithm for this task, and will be the one used for the final model, as is the algorithm which performs the best in both tasks.

6. CHAPTER

Integration of both modules

Once both modules are implemented and proven to work independently, the connection between them must be done, in order to complete the whole system, which should be structured as the image shown in the Figure 6.1. However, before completing the integration, three tasks need to be addressed:

- **Create an interface to access the ML model:** The system will have a client/server architecture, where the ML module would be the server actor, and one or more samplers from the NAM or from many modules will communicate with it. These will act as clients that send requests and receive predictions made by the ML model.
- **Communication between modules:** The clients and the servers will communicate with each other using IPC (local) or remote sockets. This way, the ML module could be installed in the same machine as the NAM, taking advantage of the multithreading architecture presented in the Section 4.6.
- **Data incompatibility between modules:** As the modules are implemented using different programming languages, some additional coding will be needed in order to adapt the differences between them and make the whole system work correctly.

6.1 The Python program

Having the model trained, the *joblib* library will be used to save it. This library allows an easy and convenient way to persist the model, as it can be exported to a file. Using the

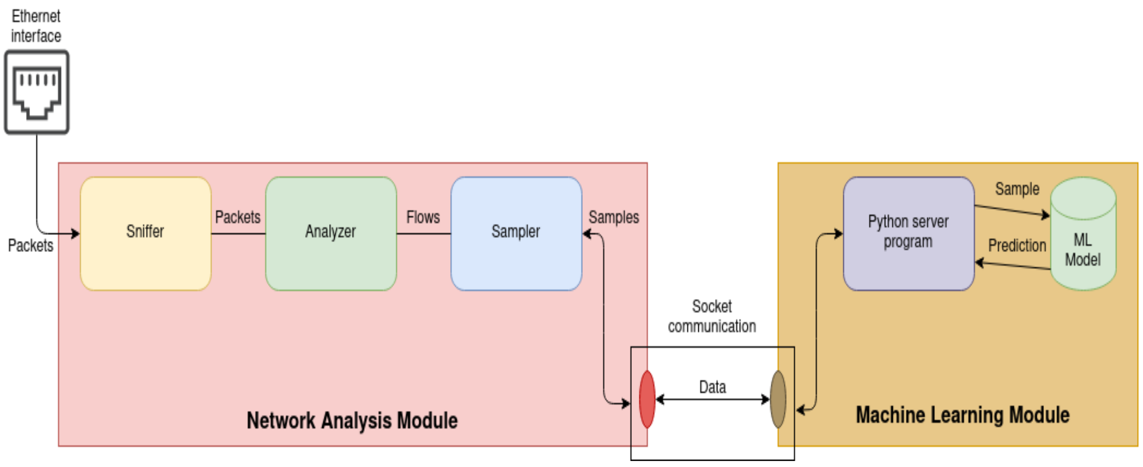


Figure 6.1: High-level representation of the modules and their interactions.

same library, the model can be imported to another program. In this case, the implementation of the server program will be carried by a simple python script, which will load the model and will act as the server for the whole system, waiting for the samples using a loop.

Every time an event is called on the NAM, the sample will be sent to this module, which will process it and feed it into the model. Now the model will make a prediction using the transmitted data, and will send it back to the client.

6.2 Communication between modules

The modules work independently from each other, however, they need to share information to accomplish the final goal of the system. The NAM is useless if the generated samples are not checked to give feedback about the status of the network and the ML module is only a running process which wastes resources if it is not fed with data. Hence, the task of communicating both of them is crucial.

In order to address the communication task, the sockets API is used, as it offers a low-level abstraction of the communication process between machines, which can be adapted to provide an easy, convenient but yet elegant solution to this problem without adding much complexity. Moreover, it does not require the installation of any new libraries, as it is included with the OS itself.

The type of socket used will be UDP and the communication will be performed using the port number 4545, although this can be changed via a configuration parameter. Needs to be mentioned, however, that the underlying socket used for the communication varies depending where these programs are the being executed. If these processes are being held on the same machine, UNIX sockets are used, as they are a more efficient method to bidirectionally transmit data within the same machine, as processes such as routing can be ignored. This is not the case when communicating with other machines, where IP sockets are needed, with all the overload that it involves.

6.3 Data incompatibilities

As mentioned before, the modules are developed in different programming languages, having each of them particularities and different technical approaches, thereby, the data structures are not represented equally.

This is the first issue found when the socket communication was being implemented, as the NAM, implemented in C, was sending to the ML module, implemented in Python, a custom C struct declared in order to be filled with the sample to be sent. The problem appeared when the struct arrived to the destination program, as python does not understand the concept of structs, hence, it was shown as an illegible sequence of bytes. This can be explained because C does not form the structs only by joining the required data types in order, in fact, C must align these data types in order to be understandable for the compiler. This is achieved by adding padding when required. The workaround for the issue was the usage of the Python library *struct*, as this offered the functions needed to describe the data typing into the sample struct. Having the layout and the description of the struct, the information was received and parsed successfully.

Now the communications were possible, the next inconvenience found was the arrangement of the unpacked data received from the other module, as it was understood as a common array by python with different datatypes and this could not be processed by the ML module. Therefore, the information of the array needed to be reshaped with the numpy [1] library to a (1,15) numpy ndarray. Having the model working properly with the received data from the socket, the final step of the program is to send back to the other module the prediction of the model.

[1] https://numpy.org/

6.4 Evaluation of the system

It was desirable for this implementation to make an evaluation of the system with real-time load, in order to analyze the predictive performance of the ML module, or parameters such as the throughput of the NAM, as this information might become useful when modelling future projects which could include this solution, although it has not been able to be done due to time constraints.

7. CHAPTER

Project management and planning

In order to fulfill correctly the tasks proposed for this project, managing the workload and the deviations properly is crucial. Correct planning will assure better results on the long run and if the estimations about the deviations are accurate, the risks will be easily managed.

The project will be divided in two main phases, according to the project structure itself. The first phase will be the management, and the second phase will be the development and documentation phase.

7.1 Description of the phases

This section will describe precisely the tasks which shape the main phases of the project. Each task is formed by a subset of simpler tasks which can be defined as **work packages**. A task would be considered completed when all the work packages associated to it are completed, therefore, a phase is complete when all its tasks are fulfilled.

7.1.1 Project management

This phase will conform the tasks associated to the management and planning of the project. The volume of work on this phase will be higher on the first hours of the project, as the workloads must be planned and distributed along time, and the deadlines must be estimated. After this initial part, the hours invested on this phase will decrease, as the management tasks will consist on small adjustments of the estimations and the monitoring and communication with the tutor, in order to receive advice.

- **Planning**: This work package is formed by the activities related to the work previous to the implementation itself. On this work package are included the division of the main goals of the project development in simpler tasks, the estimations of all the tasks of the project, and the planned meetings with the tutor. The risk management is also considered, as a risk plan will be documented.
- **Monitoring**: The activities related with the quality control of the project itself. These activities will ensure the objectives of the project are fulfilled correctly and within the defined deadlines.
- **Communication**: The main goal of this set of activities is to maintain communication with the tutor, in order to keep a constant stream of feedback from the implementation of the different tasks of the project. In fact, these meetings have also an extra value, as can be used to balance the workloads among time.

7.1.2 Project development and documentation

The actual implementation of the project is carried on this phase. Is estimated that this phase will demand the biggest time volume between all of the tasks. Once the planning is finished, the development will start, and this will continue until the closure of the project.

The development is divided in several tasks, which will be implemented in sequential order, as the tasks have dependencies from previous tasks. The main tasks are the dataset analysis, the network analysis module, the ML module and the integration of the modules, the code will be saved on a github repository[1].

Regarding the documentation itself, it will be written in parallel to the project development. Therefore, the aim is to finish each task having its documentation finished.

- **Analysis of the dataset**: One of the modules is going to be based on a machine learning model, in order to implement it and use it, a dataset will be used to feed the model with it.

[1]https://github.com/foron23/NAM

Having knowledge about the dataset itself will help to find the most relevant features for the modules.

- **Network analysis module**: This task is divided in two main subtasks, Firstly the module has to be designed. In order to accomplish this task, a set of specifications must be written, and information about the state of the art must be gathered, in order to develop a module which is useful. Once the design is finished, the task of implementing the module will be the following one to be done.

- **Machine Learning module**: Having the dataset analyzed, the next task will be to develop a model which can use the data to make accurate predictions about the network status. Therefore, that task is divided in another two subtasks, The first one being the design of the model, where many topologies and algorithms will be tried. When an effective model is found, then it will be trained with data, and it will be tuned and tweaked to maximize its performance.

- **Integration of both modules**: Having both modules developed and working, the next step is to make the connections needed for both modules so they can work together.

7.2 Estimations and deviations

The following chart shows the comparison between the estimations and the real invested hours among the different tasks which form the project.

	Planned hours	Real hours
Project Management	40	30
Planning	20	15
Monitoring	10	5
Communication	10	10
Development and Documentation	350	383
Dataset analysis	30	40
Network analysis module	110	120
Design and information gathering	30	20
Implementation	80	100
Machine Learning module	70	63
Design	10	8
Algorithm trial	40	45
Training and tuning	20	10
Integration of both modules	30	20
Documentation	70	100
Total	350	373

Table 7.1: Planned and real hours

7.2.1 Deviations

The main deviations from the planned hour estimations come from the development and documentation phase, in fact, the main deviation comes from the documentation itself, as time spent for the redaction of it was heavily underestimated, as the Table 7.1 shows. Another phases of the implementation were subject of deviation as well, however, the deviations occurred in a more balanced way, resulting in close hour counts compared to the planned hours.

Regarding the implementation, the Network Analysis module was expected to be the most extensive implementation of the whole project. Despite having that expectation, the time was overspent on it, mainly because the capture loop function worked well enough, it was needed to be more real-time alike, therefore, a more fine-grained tuning was needed to fulfill the specification, with the associated time cost to it.

Another deviation came from the dataset analysis, where the greedy algorithm based analysis was not expected, moreover, the algorithm needed further adaptation to fit the data which was being analyzed. This deviation appeared when the findings of the Sklearn based methods did not give significant results, therefore, the analysis needed to be deepened.

In a nutshell, the main deviation came from the redaction of this documentation, followed by the implementation of the Network Analysis module and the dataset analysis, although the management tasks were correctly completed and within the planned hours.

7.3 Risk Management

Most of the deviations which appear through the development of a project are related to a poor planning, or underestimating the time the tasks may need to be fulfilled. In order to prevent those risks, the approach will be to overestimate the hours needed, what gives extra room for the deviations, in case they happen.

This preemptive approach may be too conservative, therefore, inappropriate for a tightly scheduled project, however, the lack of real-life experience managing projects makes this option better compared to other options.

The current circumstances suggest that the dates of presentation may vary, leaving the deadlines of the project uncertain, which means flexibility is crucial in order to complete successfully the project.

7.4 Methodology

The design decisions about the algorithms used and the features that are integrated into the models are based on evidence extracted from papers or from preliminary results of algorithms, in order to make the choices as less arbitrary as possible.

The development of the project will be carried mainly as work done by the student, only asking help to the tutor in order to solve the doubts which appear during this process. The workloads will be distributed in 20 hours per week, which could be enough to advance the project week to week and finish it on time.

This approach has been chosen in order to cope better with a 30 hour per week internship in a company, doing this work from home, as this workload allocation still leaves some spare time to attend classes and other activities.

7.5 Workload distribution

In order to describe as accurately as possible the work done in the project, a Work Breakdown Structure (WBS) 7.1 and a Gantt diagram 7.2 will be used, in order to decompose the main objectives in more addressable tasks and in order to contextualize this tasks along the time the project is being implemented.

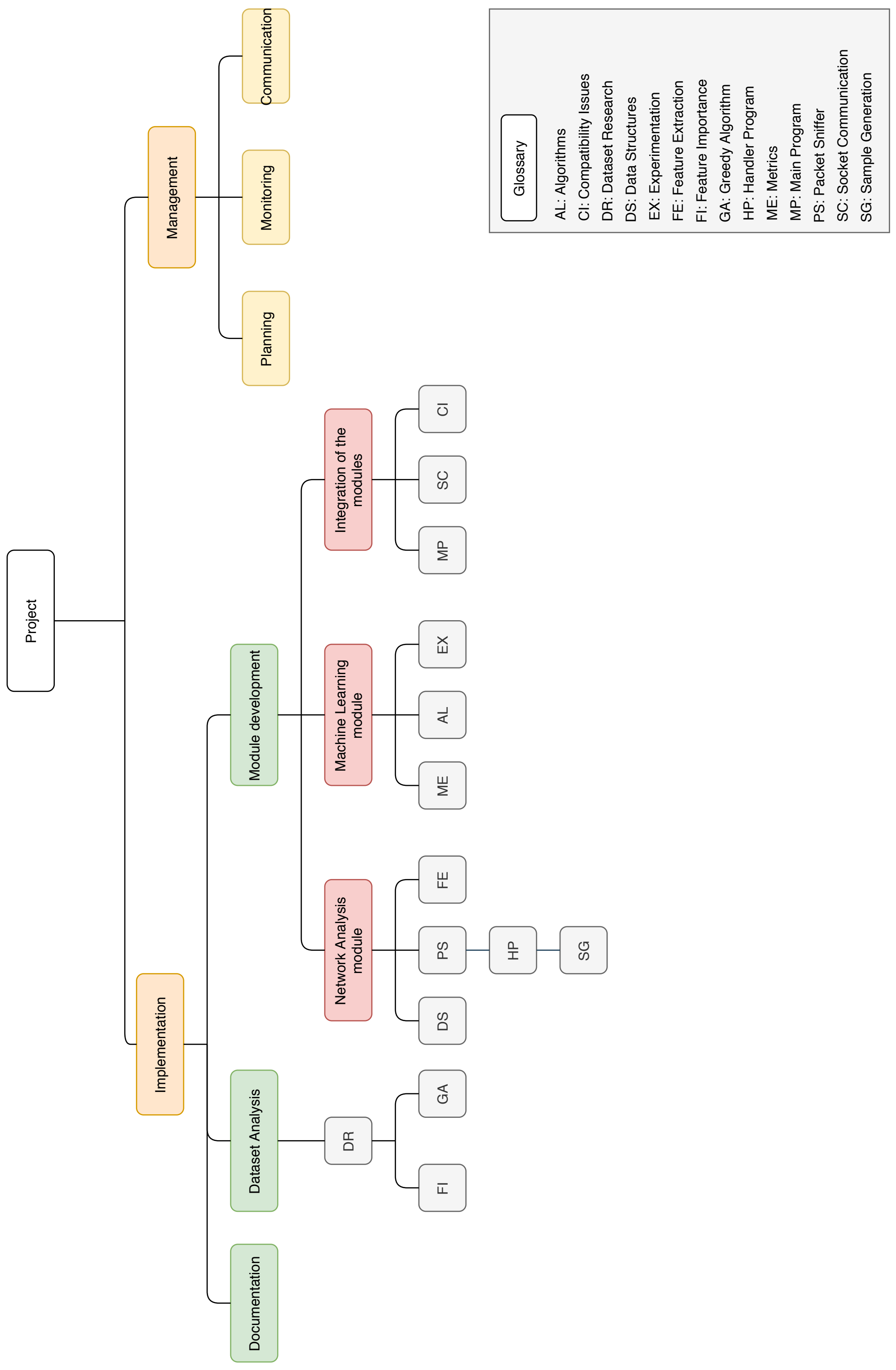


Figure 7.1: WBS diagram of the project.

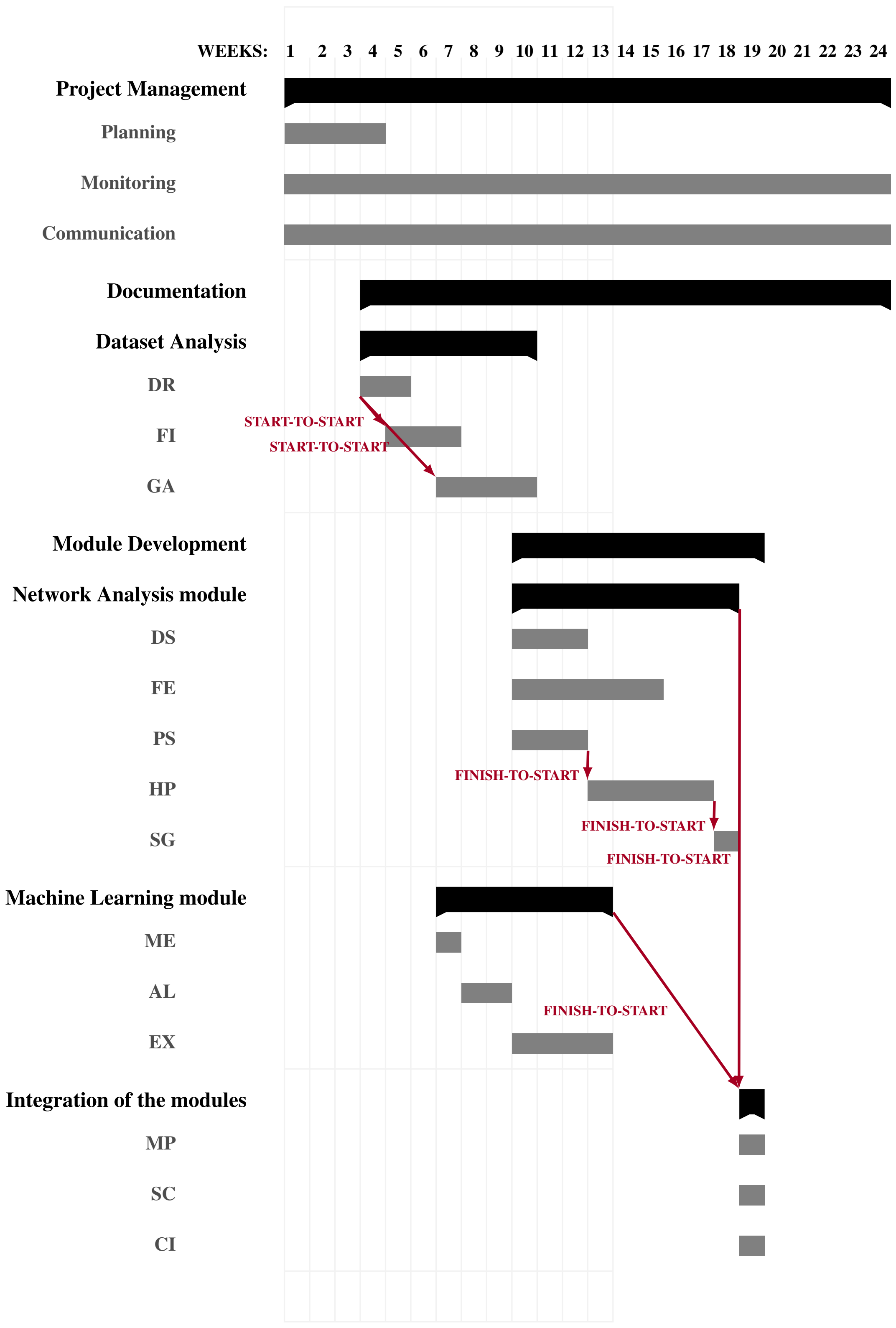


Figure 7.2: Gantt diagram of the project.

8. CHAPTER

Conclusions and future work

After finishing the implementation of the project, the potential of the project can be seen, as the whole project can be used as an extra layer on a network security system, even though to maximize its performance, it might sort necessary to train the machine learning algorithms with a dataset including samples of data extracted from traffic of the network aimed to be protected, as it is crucial to train the model in order to learn the real “boundaries” of the normal/abnormal behaviour.

Fortunately, the project provides an excellent tool in order to create a dataset, as the Network Analysis Module provides the tools needed to create data samples in real-time, thus, if this program is run on a network during a certain period of time, it could generate a sufficient amount of samples in order to build a completely custom dataset which could represent the behaviour of the network itself.

In retrospective, this project has also been an excellent opportunity to extend on the foundations given by the subjects of the degree, such as machine learning and neural networks, or the subjects of the networking department, where the basic concepts of computer networks and socket programming were taught to me, which are essential in order to develop this project.

As a possible option for future work in order to extend on this topic, it would be a useful idea to evaluate the system on a real situation, in order to model the performance of the modules. Another possible continuation of this project is the previously mentioned custom dataset generation as an approach for securing private networks.

To summarize, the whole project has been an outstanding example of how two branches of knowledge of different basis can be combined into one useful application which can improve the security in computer networks.

Bibliography

[Altman, 1992] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician. 46, Issue 3*, page 175–185.

[Andre Altmann, 2010] Andre Altmann, Laura Toloşi, O. S. . T. L. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics, Volume 26*, page 1340–1347.

[Bhuyan et al., 2014] Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys Tutorials*, 16(1):303–336.

[Breiman, 2001] Breiman, L. (2001). Random Forests. *Machine Learning 45*, pages 5–32.

[Clarke et al., 2008] Clarke, N., Furnell, S., Tjhai, G., and Papadaki, M. (2008). Investigating the problem of ids false alarms: An experimental study using snort. *International Federation for Information Processing Digital Library; Proceedings of The Ifip Tc 11 23rd International Information Security Conference ;*, 278.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function mathematics of control. *Signals, and Systems, 2 Issue 4*, pages 303–314.

[Denning, 1987] Denning, D. E. (1987). An intrusion-detection model. *IEEE transactions on software engineering*, 13(2):222–232.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Kira and Rendell, 1992] Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. pages 249 – 256.

[Meena and Choudhary, 2017] Meena, G. and Choudhary, R. R. (2017). A review paper on ids classification using kdd 99 and nsl kdd dataset in weka. pages 553–558.

[Mell et al., 2001] Mell, P., Hu, V., Lippmann, R., Haines, J., and Zissman, M. (2001). An overview of issues in testing intrusion detection systems1.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition.

[Moustafa and Slay, 2015] Moustafa, N. and Slay, J. (2015). UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6.

[Rokach, 2010] Rokach, L. (2010). Ensemble-based classifiers. *Artif. Intell. Rev.*, 33:1–39.

[Rokach and Maimon, 2008] Rokach, L. and Maimon, O. (2008). *Data mining with decision trees. Theory and applications*, volume 69.

[Rumelhart and Williams, 1986] Rumelhart, David E., G. E. H. and Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundation.*

[Shon and Moon, 2007] Shon, T. and Moon, J. (2007). A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799 – 3821.

[Tavallaee et al., 2009] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. pages 1–6.