



République Algérienne Démocratique et Populaire



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique  
Département Informatique

Mémoire de Master

Filière : Informatique

Spécialité : Réseaux et Systèmes Distribués

---

## **Thème**

**UNE APPROCHE DE DETECTION DE BOTNETS DANS  
LE RESEAU DEFINI PAR LOGICIEL (SDN).**

---

Sujet Proposé par :

**Mme BOUGHACI Dalila**

Soutenu le : ../../....

Présenté par

**BESSAI Mohamed Cherif**

**GUENFAF Youcef**

Devant le jury composé de :

**M..... Président (e)**

**M..... Membre**

Binôme N° : 098/ 2020

# Résumé

La nouvelle tendance des réseaux Software Defined Networks est le nouveau sujet tendance dans le domaine des communications informatiques. Ces réseaux apportent une grande flexibilité et capacité de passage à l'échelle afin de répondre aux exigences des nouvelles technologies (Cloud Computing, IoT, Big Data...). Par ce travail nous avons étudié la problématique de la sécurité dans ce type de réseaux notamment face aux attaques Botnets. Nous avons proposé un système de détection de ces intrusions en se basant sur l'apprentissage en profondeur (Deep Learning). Nous avons ensuite simulé notre solution sur un cas pratique de SDN grâce à l'environnement Mininet. Notre système a montré de bonnes performances par rapport aux solutions qui existent dans la littérature.

Mot clefs: Software Defined Networks (SDN), Botnet, Détection D'Attaques, Apprentissage Automatique, Apprentissage en profondeur, Hogzilla Dataset, Mininet

# Abstract

Software Defined Networks are the newest hot topic in the world of computer communications, especially for the kind of flexibility and scalability they bring to the table in terms of meeting the requirements of new technologies such as Cloud Computing, IoT, Big Data... Through this work, we have studied the security challenges in this types of networks, in particular the botnet attacks. We proposed a system developed around the use of Deep Learning, before performing a simulation on a virtual SDN with the Mininet environment. Our solution showed the best performances compared to similar existing solutions.

Keywords: Software Defined Networks (SDN), Botnet, Attack Detection, Machine Learning, Deep Learning, Hogzilla Dataset, Mininet.

# Table des matières :

## Chapitre 1 : Software-Defined Networks

Chapitre 1 : Software-Defined Networks.....	II
1.1 Introduction : .....	3
1.2 Définition :.....	3
1.3 Caractéristiques :.....	3
1.3.1 La séparation des plans : .....	3
1.3.2 La centralisation du contrôle : .....	4
1.3.3 L'automatisation et la virtualisation du réseau : .....	4
1.3.4 L'ouverture : .....	4
1.4 Architecture : .....	5
1.5 Contrôleur SDN : .....	7
1.6 Le Protocole Openflow : .....	8
1.6.1 Les Types de messages entre les OpenFlow switchs et le contrôleur : .....	9
1.7 Les menaces de sécurité dans le SDN : .....	9
1.8 Conclusion : .....	11

## Chapitre 2 : Botnets, détection des attaques et machine Learning

2.1 Introduction : .....	12
2.2 Définition : .....	12
2.3 Le cycle de vie d'un Botnet : .....	12
2.4 L'architecture d'un Botnet : .....	14
2.4.1 Botnets Client-serveur : .....	14
2.4.2 P2P Botnets : .....	15
2.5 Les différents types d'attaques Botnet : .....	16
2.6 Les méthodes de détection de Botnet : .....	16
2.6.1 Détection basée sur les honeynets .....	17
2.6.2 Détection basée sur les IDS (Système de Détection des Intrusions) : .....	17
2.7 Machine Learning (apprentissage automatique) : .....	19

2.7.1 Définition :	19
2.7.2 Types d'apprentissage :	19
2.7.3 Les modèles de machine Learning :	20
2.8 Deep Learning (l'apprentissage profond) et réseau de neurones :	25
2.8.1 Définition de l'apprentissage profond :	25
2.8.2 Domaines d'application d'apprentissage profond :	26
2.8.3 Réseaux de neurones artificiels :	26
2.8.4 Réseau de neurones monocouche (Le perceptron) :	28
2.8.5 Réseau de neurones multicouche :	28
2.8.6 Les différents types de fonction d'activation :	29
2.8.7 Fonctions de propagation et rétropropagation :	30
2.9 Comparaison d'apprentissage profond et l'apprentissage automatique :	31
2.10 Travaux antérieurs :	31
2.11 Conclusion :	33

## Chapitre 3 : Approche proposée pour la détection des botnets dans réseau SDN

3.1 Introduction :	34
3.2 Architecture générale de notre système :	34
3.2.1 Collecteur du flux :	35
3.2.2 Module d'extraction des caractéristiques :	37
3.2.3 Module de classification des flux :	39
3.3 Conclusion :	40

## Chapitre 4 : Validation sur HogZilla dataset

4.1 Introduction :	41
4.2 Le jeu de données Hogzilla (Hogzilla dataset) :	41
4.3 La sélection des attributs (Feature Selection) :	42
4.4 Adaptation du réseau de neurones au jeu de données HogZilla :	46
4.5 Langage de programmation et bibliothèques utilisés :	48
4.5.1 Python :	48
4.5.2 Tensorflow :	48

4.5.3 Keras :	48
4.5.4 NumPy :	48
4.5.5 Scikit-learn :	48
4.5.6 Pandas :	49
4.6 Outils et logiciels de développements :	49
4.6.1 Jupyter Notebook :	49
4.7 Résultats numériques :	49
4.7.1 Matrice de confusion (Confusion matrix) :	49
4.7.2 Métriques d'évaluation :	50
4.7.3 Evaluation des performances :	51
4.8 Conclusion :	53

## Chapitre 5 : Simulation Mininet et étude Comparative des différents résultats

5.1 Introduction :	54
5.2 Environnement de travail :	54
5.2.1 Environnement matériel :	54
5.2.2 Environnement logiciel :	54
5.3 La Simulation des attaques Botnet dans un environnement SDN :	56
5.3.1 Topologie Mininet :	56
5.3.2 Les attaques DDOS :	57
5.4 Etude comparative des différents résultats :	63
5.4.1 Comparaison des différentes configurations :	63
5.4.2 Comparaison des différents algorithmes de Machine Learning et Deep Learning :	66
5.4.3 Comparaison de notre approche et travaux antérieurs :	67
5.5 Conclusion :	68
<b>Bibliographie et webographie</b>	<b>70</b>

# Liste des tableaux :

Tableau 1.1 : comparaison entre les caractéristiques de l’architecture SDN et l’architecture classique .....	5
Tableau 2.1 : Comparaison des différentes techniques de détection de Botnet .....	19
Tableau 2.2 : Les différents type de fonction d’activation .....	29
Tableau 2.3 : comparaison entre l’apprentissage automatique et l’apprentissage profond .....	31
Tableau 3.1: Tableau des différentes statistiques que le module de collection de flux peut extraire à partir de la table de flux d’un switch OpenFlow.....	37
Tableau 4.1 : Les attributs choisis et leurs scores attribués par la formule chi-square.	43
Tableau 4.2 : Les attributs extraits du contrôleur et leurs équivalents dans HogZilla ..	44
Tableau 4.3 : les attributs HogZilla calculés et dérivés dans un environnement SDN .	45
Tableau 4.4 : La matrice de confusion.....	50
Tableau 4.5 : Matrice de confusion avec Botnet est la classe positive et trafic normal est la classe négative .....	52
Tableau 4.6 : Les résultats obtenus dès la validation de notre approche au HogZilla dataset.....	52
Tableau 5.1 : Les résultats obtenus par les différents nombres d’époques testés.....	63
Tableau 5.2 : Les résultats obtenus par les différentes tailles du lot testées.....	64
Tableau 5.3 : Les résultats obtenus par les différentes configurations testées .....	65
Tableau 5.4 : Tableau comparatif des différents résultats obtenus par les algorithmes d’apprentissage automatique et profond dans un environnement d’attaques .....	66
Tableau 5.5 : Tableau comparatif des différents résultats obtenus par les algorithmes d’apprentissage automatique et profond dans un environnement ordinaire .....	66
Tableau 5.6 : Tableau comparatif des différents travaux.....	67

# Liste des Figures :

Figure 1.1 : Architecture SDN et architecture classique de réseau [1] .....	5
Figure 1.2 : L'architecture de SDN en couches .....	7
Figure 1.3 : Architecture d'un contrôleur SDN et ses interfaces .....	8
Figure 2.1 : cycle de vie d'un Botnet .....	14
Figure 2.2 : Architecture client-serveur des Botnets .....	15
Figure 2.3 : Architecture Botnet P2P .....	15
Figure 2.4 : Classification des différentes méthodes de détection des Botnets .....	17
Figure 2.5 : Exemple d'un arbre de décision .....	21
Figure 2.6 : La courbe qui définit la fonction logistique (fonction sigmoïde) .....	22
Figure 2.7 : Structure générale d'un neurone .....	27
Figure 2.8 : la représentation de système nerveux dans le réseau de neurones artificiel .....	27
Figure 2.9 : Réseau de neurones monocouche .....	28
Figure 2.10 : réseau de neurones multicouche avec deux couches cachées .....	29
Figure 3.1 : Présentation de l'architecture de système de détection des Botnets .....	35
Figure 3.2 : Fonctionnement de collecteur du flux.....	36
Figure 3.3 : Processus d'apprentissage et classification des flux.....	40
Figure 5.1 : La topologie construite avec l'outil Mininet .....	56
Figure 5.2 : Exemple d'attaque SYN FLOOD .....	57
Figure 5.3 : La création de topologie SDN par outil Mininet.....	58
Figure 5.4 : Les six instances des six hôtes d'émulateur de terminal Xterm.....	59
Figure 5.5 : L'établissement de connexion entre le Bot maître et les esclaves .....	60
Figure 5.6 : Démarrage du serveur victime par l'outil IPERF .....	60
Figure 5.7 : Inondation du serveur UDP par les esclaves .....	61
Figure 5.8 : Initialisation de contrôleur RYU avec le script de détection .....	62
Figure 5.9 : Un aperçu sur l'application de détection des attaques Botnet.....	62

# Introduction Générale

Dans une société où technologie et communication ont pris une place primordiale dans notre vie de tous les jours, personne ne peut nier l'impact du partage de l'information et de l'infrastructure dont cela dépend. Et comme outil indispensable de cette infrastructure, nous ne pouvons douter que l'avenir des réseaux informatiques soit de grandir et de se développer. Cet avenir est pour une bonne part liée aux techniques et aux supports de communication utilisés dans les réseaux. Bien que la technologie actuelle permette d'accroître les volumes et les vitesses de transfert des données tout en diminuant les coûts, cet accroissement de la taille des réseaux ainsi que l'avènement de nouvelles technologies telles que le Cloud Computing, le Big Data et l'IoT présentent quelques problèmes. En effet, les réseaux informatiques ont toujours été configurés et opérés par des humains alors que la complexité des technologies, des services mis en jeu, et la rapidité de livraison de ces services ne fait qu'augmenter. Aujourd'hui flexibilité, agilité et rapidité, sont les maîtres mots client, nécessitant de rendre le réseau plus programmable afin de continuer dans le développement de ce dernier.

C'est dans ce contexte que la technologie Software Defined Networking (SDN) est proposée comme l'une des meilleures solutions qui répondent aux exigences de passage à l'échelle et changements dynamiques posés par les réseaux actuels. Cette nouvelle technologie possède la capacité d'être programmable, configurable et gérable grâce à un plan de contrôle centralisé. La sécurité de ces réseaux est la première priorité des chercheurs dans le domaine vu le rôle principal qu'ils joueront dans l'avenir et le développement des réseaux informatiques.

En effet, de nombreux vecteurs d'attaques empêchent encore cette technologie d'être déployée. Parmi ces attaques, les Botnets peuvent être efficacement utilisés pour causer de sérieux dégâts dans ce type d'architecture. Que ce soit des attaques de déni de service, des usurpations d'identité ou encore des extractions de cryptomonnaie, les SDNs exigent une protection contre ce genre de menace.

L'objectif de ce travail est de proposer un système de détection des attaques Botnet qui contribue à l'avancement des recherches dans cette problématique. Pour cela, nous devons tout d'abord :

- Etudier les concepts du réseau SDN
- Etudier les limitations et problèmes de sécurité posés par les attaques Botnet dans les SDNs
- Etudier les solutions et systèmes de détection existants
- Proposer une solution de détection d'intrusions Botnet dans les SDNs
- Valider et tester la solution proposée



Afin de mener à bien ce projet et de présenter notre travail convenablement, nous avons organisé notre mémoire en cinq chapitres. Le premier chapitre présentera les différents concepts de l'environnement SDN ainsi que leurs caractéristiques et leurs architectures. Le deuxième chapitre se focalisera sur les problèmes de sécurité apportés par les Botnets ainsi que les solutions de sécurité déjà proposées. Nous consacrerons aussi une partie du chapitre à se familiariser avec les techniques d'apprentissage automatique et plus précisément l'apprentissage profond. Dans le troisième chapitre nous présenterons notre conception d'une solution permettant la détection des attaques Botnet. Dans le quatrième chapitre, nous validerons notre solution sur le dataset Hogzilla après avoir présenté ce dernier. Nous conclurons dans le dernier chapitre par l'implémentation de notre solution ainsi que la comparaison des performances avec d'autres solutions. Le tout suivi d'une conclusion générale.

# Chapitre 1: Software-Defined Networks

## 1.1 Introduction :

Le SDN est une technologie encore récente qui a déjà commencé à remplacer les modèles traditionnels de networking tels que nous les connaissons. Il offre des avantages considérables en termes de flexibilité et d'adaptation pour subvenir aux besoins des nouvelles tendances réseaux et IoT tels que le cloud, la mobilité, le social networking et la vidéo.

Alors qu'est-ce que le SDN ? Quel est son principe de fonctionnement ? Et quel avantage apporte-t-il par rapport aux réseaux classiques ?

C'est à ces questions que nous essayerons de répondre tout au long de ce chapitre.

## 1.2 Définition :

Software-Defined Network (SDN) est une architecture émergente qui sépare les fonctions de contrôle (control plane) et de transfert de données (data plane), permettant de rendre le contrôle plus programmable et d'offrir une vue abstraite sur l'infrastructure du réseau aux différentes applications et services [1].

## 1.3 Caractéristiques :

Les SDNs se caractérisent par 4 traits fondamentaux : la séparation des plans, la centralisation du contrôle, l'automatisation et la virtualisation du réseau, et l'ouverture [2].

### 1.3.1 La séparation des plans :

La première caractéristique fondamentale du SDN est la séparation des plans de données et de contrôle. La fonctionnalité de transfert, y compris la logique et les tables permettant de choisir la manière de traiter les paquets entrants, en fonction de caractéristiques telles que l'adresse MAC, l'adresse IP et l'ID VLAN, résident dans le plan de données.

La logique et les algorithmes utilisés pour programmer le plan de données résident dans le plan de contrôle. Le plan de contrôle détermine comment la logique de transfert des paquets dans le plan de données doit être programmée ou configurée.

### 1.3.2 La centralisation du contrôle :

L'intelligence du réseau est centralisée via les contrôleurs SDN qui dictent les stratégies réseaux. Le contrôleur fournit des instructions aux dispositifs réseaux, afin de leur permettre de prendre des décisions rapides sur la manière de traiter et acheminer les paquets entrants.

### 1.3.3 L'automatisation et la virtualisation du réseau :

Les solutions SDN permettent d'ajouter de nouveaux composants ou de configurer l'existant grâce à des interfaces programmables ou à l'utilisation de scripts.

Les administrateurs peuvent désormais ajuster dynamiquement et automatiquement les flux de trafic à l'échelle du réseau pour répondre à des besoins fluctuants.

### 1.3.4 L'ouverture :

Les protocoles et interfaces utilisés sont standards et non liés à un fournisseur. Ceci afin de garantir aux institutions de recherche et aux entrepreneurs la possibilité de tester et expérimenter de nouvelles méthodes plus innovatrices en matière d'opérations réseaux. De plus, des interfaces standards permettent d'incorporer des équipements de différents fournisseurs, offrant ainsi un environnement plus compétitif et donc une réduction dans les prix pour les consommateurs des équipements réseaux.

Caractéristiques	SDN	Architecture classique
Programmable	Oui	Non
Contrôle centralisé	Oui	Non
Configuration sujette aux erreurs	Non	Oui
Complexité de contrôle	Non	Oui
Réseau flexible	Oui	Non
Performance améliorée	Oui	Non
Implémentation facile	Oui	Non

Tableau 1.1 : comparaison entre les caractéristiques de l'architecture SDN et l'architecture classique [1]

#### 1.4 Architecture :

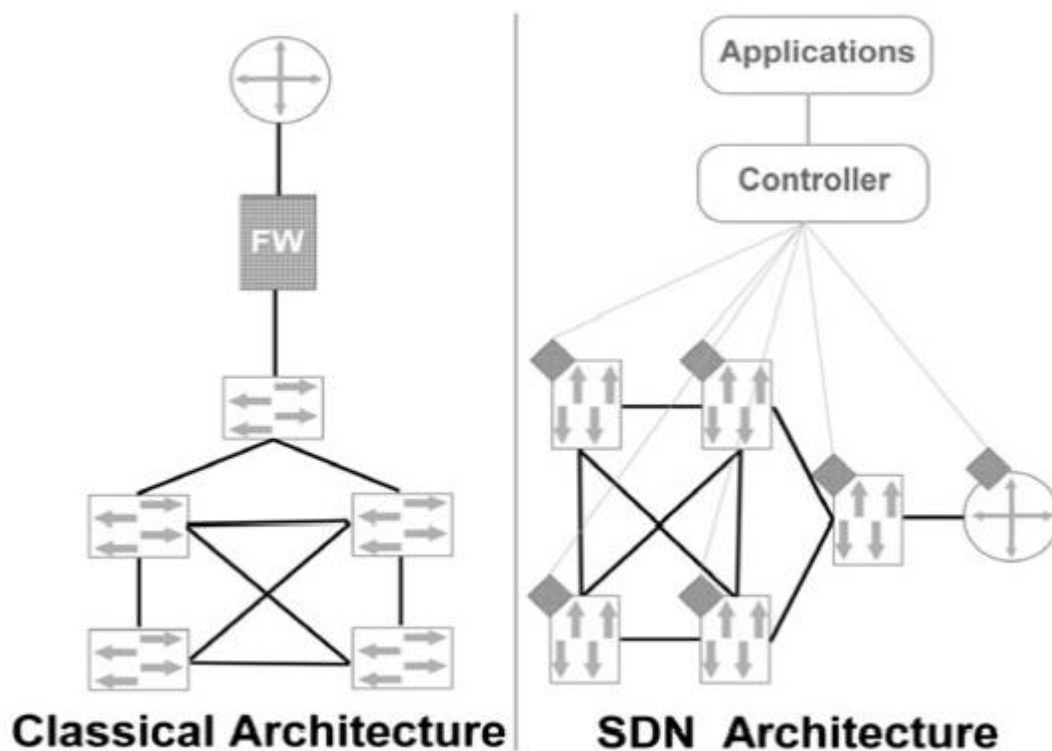


Figure 1.1 : Architecture SDN et architecture classique de réseau [1]

L'architecture de base de SDN est divisée en trois couches qui communiquent entre elles à travers des APIs :

**Couche d'application (application plane) :** La couche supérieure de l'architecture SDN est la couche d'application qui propose différents services tels que les pare-feux, le contrôle d'accès, IDS / IPS, la qualité de service, le routage, le service proxy...etc

Les applications SDN sont des programmes qui communiquent explicitement, directement et par programmation leurs exigences réseau et le comportement réseau souhaité au contrôleur SDN via des interfaces NBI (Northbound API) [1].

**Couche de contrôle (Control plane) :** Le contrôleur est le composant principal responsable de l'établissement des tableaux de flux et des politiques de traitement des données ainsi que l'abstraction de la complexité du réseau et la collecte des informations réseaux via l'API Southbound, le contrôleur maintient aussi une vue globale à jour du réseau [1].

Le contrôleur SDN est une entité logiquement centralisée chargée (i) de traduire les exigences de la couche d'application SDN vers la couche de données SDN et (ii) de fournir aux applications SDN une vue abstraite du réseau (qui peut inclure des statistiques et des événements) [3].

**Couche de données (Data plane) :** La couche la plus basse, fournit des périphériques de réseau tels que des commutateurs physiques / virtuels, des routeurs et des points d'accès et elle est responsable du transfert, de la fragmentation et du réassemblage de données [1].

### **Les APIs : Interface de programmation (Application Program Interface)**

**Northbound API :** Les Northbound APIs de réseau SDN sont des interfaces entre la couche application SDN et les contrôleurs (couche contrôle) SDN et fournissent généralement une vue abstraite du réseau permettant aux logiciels de la couche application de fournir les algorithmes et les protocoles qui peuvent gérer le réseau efficacement. Ces applications peuvent apporter rapidement et dynamiquement des modifications au réseau en fonction des besoins [2].

**Southbound API :** Le Southbound API est l'interface définie entre un contrôleur SDN et la couche de données SDN, qui fournit au moins (i) le contrôle de toutes les opérations de transfert, (ii) la génération de rapports statistiques et (iii) notification d'événements [3].

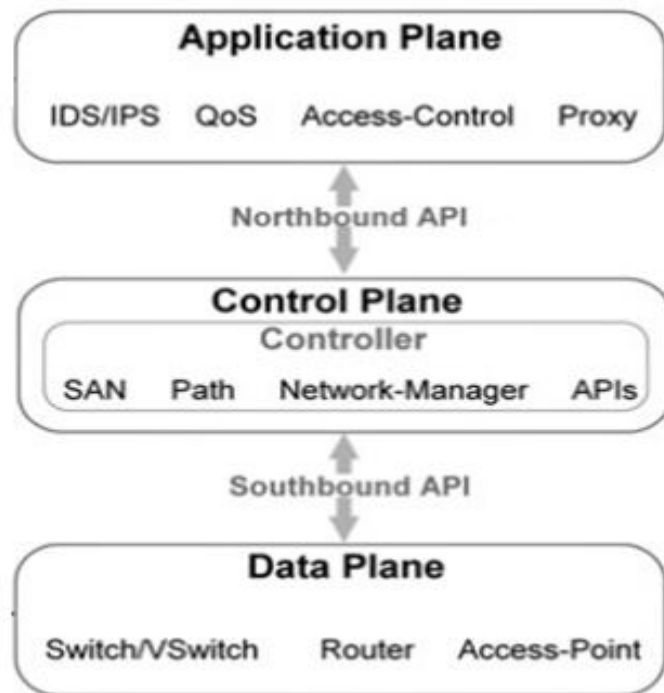


Figure 1.2 : L'architecture de SDN en couches [1]

## 1.5 Contrôleur SDN :

La fonctionnalité fournie par le contrôleur SDN peut être considérée comme un système d'exploitation réseau (Network Operating System **NOS**) [5].

Un NOS est une plateforme logicielle fournit aux développeurs des services essentiels, des interfaces de programmation d'applications (API) communes pour gérer le réseau d'une manière centralisée [5].

Les fonctions d'un contrôleur, permettent aux développeurs de définir des politiques réseaux et de gérer des réseaux sans se soucier des détails de caractéristiques des périphériques réseaux, qui peuvent être hétérogènes et dynamiques [5].

Un certain nombre d'initiatives, à la fois commerciales et open source, ont abouti à des implémentations de contrôleurs SDN. La liste suivante en décrit quelques-uns :

**OpenDaylight** (ODL) : Une plate-forme open source SDN, écrite en Java. OpenDaylight a été fondée par Cisco et IBM. Elle peut être implémenté comme un contrôleur centralisé unique, mais permet aux contrôleurs d'être distribués là où une ou plusieurs instances peuvent s'exécuter sur un ou plusieurs serveurs en cluster dans le réseau [4].

**Open Network Operating System (ONOS)** : Il s'agit d'un NOS SDN open source qui est pris en charge par l'Open Networking Foundation (ONF) et qui est conçu pour être utilisé en tant que contrôleur distribué. Il offre une haute disponibilité grâce au clustering [4].

**POX** : Un contrôleur OpenFlow open source qui a été implémenté par un certain nombre de développeurs et d'ingénieurs SDN. Il fournit également une interface graphique (GUI) et est écrit en Python [4].

**Ryu**: Une infrastructure logicielle (**Framework**) SDN basée sur des composants open source développé par NTT Labs. Ryu est open source et entièrement développé en Python [4].

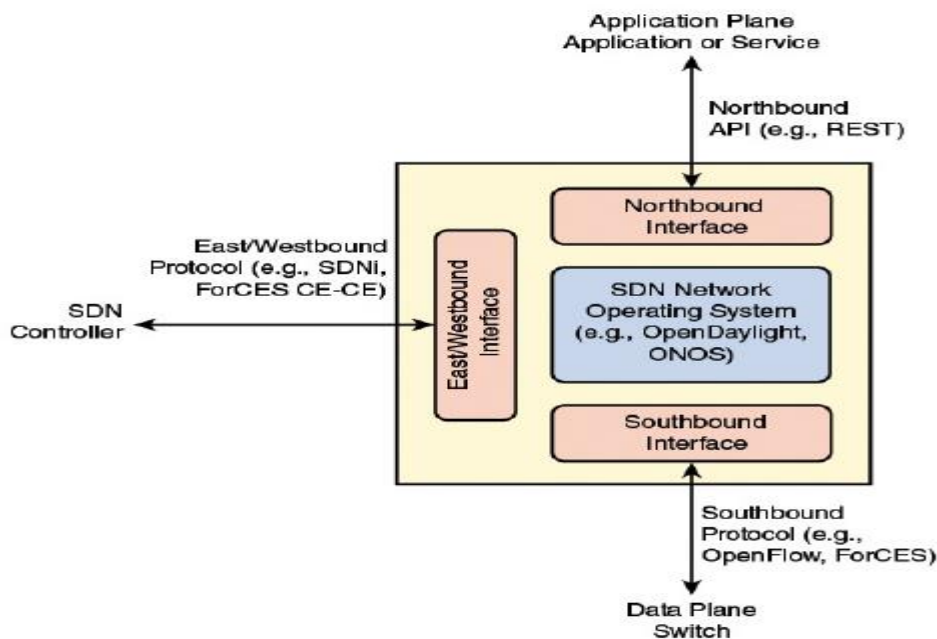


Figure 1.3 : Architecture d'un contrôleur SDN et ses interfaces [5]

## 1.6 Le Protocole OpenFlow :

OpenFlow est un protocole qui fournit une manière standardisée de gérer le trafic et décrit comment un contrôleur communique avec des périphériques réseaux tels que des commutateurs et des routeurs. Ces périphériques se composent de deux composants logiques : **un tableau de flux** (flow table) qui définit comment traiter et transférer les paquets au sein du réseau et une Interface de programmation d'application (API) OpenFlow qui gère les échanges entre le commutateur / routeur et le contrôleur [1].

### 1.6.1 Les Types de messages entre les OpenFlow switches et le contrôleur :

Il existe 3 types de messages entre le commutateur et le contrôleur [6] :

#### Contrôleur-à-commutateur :

Des messages initiés par le contrôleur, soit pour demander les fonctionnalités prises en charge du commutateur (**message FEATURES\_REQUEST**), soit pour mettre à jour la configuration du commutateur (**SET\_CONFIG**) ou sa table de flux (**PACKET\_OUT** qui est la réponse à une requête **PACKET\_IN** puis un message **FLOW\_MOD** qui modifie la table des flux) et enfin si le contrôleur a besoin des statistiques de la table du flux d'un commutateur (**message STATS\_REQUEST**).

#### Messages asynchrones :

Les messages sont envoyés du commutateur au contrôleur afin de mettre à jour le commutateur en cas de manque d'une entrée pour un flux dans la table des flux (**flow table**) ses messages sont appelés **PACKET\_IN**.

Un autre type des messages **STATS\_REPLY** qui sont des messages réponses aux requêtes **STATS\_REQUEST** qui contient les statistiques concernant les flux installés dans la table de flux.

#### Messages symétriques :

Les messages sont initiés par le contrôleur et le switch. Des exemples de ces messages incluent l'échange des messages **HELLO**, utilisés dans la prise de contact initial entre le commutateur et le contrôleur pour se mettre en accord sur la version de Openflow utilisée ou des messages **ECHO** envoyés périodiquement pour confirmer la disponibilité de la connexion entre le contrôleur et le commutateur.

## 1.7 Les menaces de sécurité dans le SDN :

La sécurité dans les SDNs consiste à protéger les informations contre le vol ou l'endommagement du matériel et logiciel ainsi que l'interruption des services. La sécurisation du SDN englobe la sécurité physique du matériel, ainsi que la prévention des menaces logicielles pouvant provenir du réseau ou des données [1].



Alors que le SDN fournit une nouvelle conception de réseau permettant une large gamme d'applications et de services réseau d'être utilisés, les problèmes de sécurité sont devenus un pilier important, car la sécurité n'est pas encore une fonction intégrée dans l'architecture SDN. Basée sur les technologies existantes (par exemple, routeurs, commutateurs, serveurs, applications, etc.), elle hérite systématiquement des problèmes de sécurité qui lui sont liés (problèmes de réseaux traditionnels) [1].

Comme le SDN s'appuie principalement sur divers logiciels de la couche application pour interagir avec l'infrastructure sous-jacente, cela présente sûrement des inconvénients majeurs car les vulnérabilités du code peuvent avoir un impact sérieux sur la sécurité [1].

De plus, en centralisant le contrôleur pour la gestion et le contrôle, cela entraînera la création d'un point de défaillance. Les contrôleurs pourraient être menacés de différents côtés (c'est-à-dire les interfaces API Northbound et Southbound) [1].

La couche de données est également une cible potentielle pour les attaquants. En raison des échanges entre les plans de contrôle et de données, un attaquant pourrait perturber le flux de données en inondant les commutateurs (Flooding) ou alternant la communication. Le protocole OpenFlow chargé d'établir les communications entre les deux couches(plans) peut également être vulnérable à certaines attaques [1].

Les vulnérabilités SDN peuvent être réparties en cinq principaux axes [1] :

**Couche d'application** : il s'agit de vulnérabilités liées au logiciel qui pourraient être exploitées par injection de code.

**Couche de contrôle** : cela inclut les vulnérabilités associées au contrôleur.

**API** : la couche de contrôle a deux types d'API : vertical (c'est-à-dire en direction sud et nord) et horizontal (c'est-à-dire en direction est, ouest pour lier les différents contrôleurs). Des recherches ont principalement mis en évidence le manque de sécurité dans les processus de communication entre les applications OpenFlow et le plan de contrôle ou entre différents contrôleurs.

**Les protocoles** : dans une architecture SDN basée sur OpenFlow, le protocole OpenFlow peut être exposé à diverses menaces. Cela est dû au fait que les dernières versions implémentent le protocole TLS qui établit une connexion sécurisée entre le switch et le contrôleur, cependant son utilisation est optionnelle et reste donc ignorée par beaucoup d'utilisateurs.

**Couche de données** : les vulnérabilités liées aux composants de la couche de données (par exemple, les commutateurs et les routeurs) pourraient engendrer différents types d'attaques tels que les modifications de règles de flux et le Flooding (inondation).

## 1.8 Conclusion :

Dans ce chapitre nous avons présenté quelques notions sur les réseaux SDNs, leurs caractéristiques et l'architecture de leur déploiement. Par ailleurs, nous avons mentionné les menaces de sécurité auxquelles est exposée ce genre d'infrastructures. Dans le prochain chapitre nous nous concentrerons sur les attaques Botnets dans ce réseau ainsi les méthodes de détection de ces dernières.

# Chapitre 2 : Botnets, détection des attaques et machine Learning

## 2.1 Introduction :

Les Botnets sont une forme de cyber menace responsables d'attaques de déni de services (DDoS), la délivrance suspicieuse de demande de rançons comme les ransomware, ou encore la dissémination de spam qui peuvent être utilisés pour le phishing. Les Botnets sont étroitement liés aux IoT (Internet of Things) et plus particulièrement aux machines IoT, qui, quand elles sont compromises, deviennent une partie d'un Botnet.

Récemment le nombre de machines connectées à Internet augmente d'un tiers chaque année, allant de plus de 5 milliards en 2015 à plus de 20 milliards d'ici 2021. C'est incroyable accroissement des machines IoT ainsi que la relation étroite entre les Botnets et les autres attaques place ces derniers comme étant l'une des cybers menaces les plus importantes dans le monde, placée à la 8<sup>ème</sup> position par l'agence de réseau et de sécurité informatique de l'union européenne (ENISA).

Dans ce chapitre, nous étudierons le problème de Botnet dans les réseaux SDN. Nous présenterons ensuite quelques méthodes et modèles de détection des attaques Botnet et nous conclurons par une présentation des algorithmes de l'apprentissage automatique et ceux de l'apprentissage profond qui ont été récemment appliquées pour détecter ces attaques à cause de leur fiabilité avant de citer quelques travaux liés à ce sujet.

## 2.2 Définition :

Un Botnet est un réseau constitué de plusieurs machines connectées entre elles, chacune exécutant un script qui réalise une simple tâche de façon répétitive. Un Botnet typique est généralement constitué d'un bot serveur (un serveur IRC par exemple) appelé maître et un ou plusieurs Bots clients appelés esclaves [7].

## 2.3 Le cycle de vie d'un Botnet :

Les Botnets suivent des étapes similaires tout au long de leur durée de vie, la durée de vie d'un Botnet comprend trois étapes principales, comme suit [8] :

**Étape 1 - étape de recrutement :** la formation du réseau Botnet commence par recruter autant de machines vulnérables que possible pour faire partie de ce réseau. Cela se fait en infectant les machines victimes avec le code bot (un script malveillant) en utilisant différents mécanismes. L'un des mécanismes est la propagation de vers (**Worm**) pour propager des logiciels malveillants de Botnets. Une machine infectée a la capacité de rechercher d'autres machines vulnérables grâce à une analyse active des trous de vulnérabilités connues.

Social Engineering est aussi un mécanisme puissant qui est utilisé par les Botmasters (attaquants principaux ou bien attaquants maitres) pour convaincre les utilisateurs finaux de télécharger des fichiers contenant le script malveillant

**Étape 2 – étape C&C (Command and Control) :** Le Botmaster contrôle les machines infectées (bots) via un canal C&C. L'architecture du Botnet dépend de l'implémentation du canal C&C. Dans les Botnets centralisés, le Botmaster contrôle son Botnet via un serveur central appelé serveur C&C. Dans les Botnets P2P, il n'y a pas de serveur central entre le Botmaster et les machines Botnet. Par conséquent, le Botmaster communique directement avec un petit sous-ensemble de machines de Botnet.

Ces machines du sous-ensemble servent de boîtes aux lettres entre le Botmaster et d'autres machines de Botnet.

**Étape 3 - Étape d'activité du Botnet :** L'activité du Botnet représente l'ensemble des actions et des attaques (par exemple, DDoS, analyse, etc.) qui sont effectuées par les bots en réponse aux commandes émises par le Botmaster

Ces étapes peuvent être vues comme un cycle de vie est résumé dans le schéma suivant :

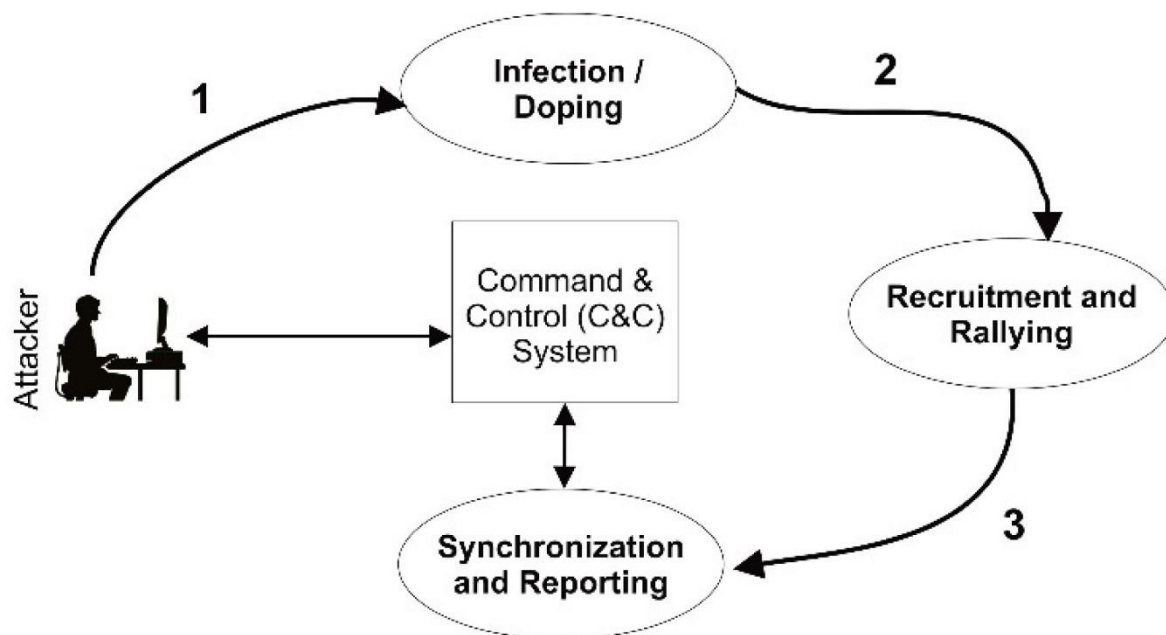


Figure 2.1 : cycle de vie d'un Botnet [9]

## 2.4 L'architecture d'un Botnet :

### 2.4.1 Botnets Client-serveur :

La plupart des Botnets (par exemple : sdbot, agobot, GTbot) apparus au début de l'ère des Botnets ont adopté une architecture centralisée. Dans cette architecture, le Botmaster maintient un serveur central (serveur C&C) qui communique avec les bots. Les bots attendent les commandes du serveur central. De plus, les hôtes (bots) nouvellement compromis se connectent au serveur et communiquent leurs informations. Le serveur supervise l'état des bots et envoie des commandes à exécuter.

Dans les Botnets centralisés, le canal C&C peut être implémenté à l'aide de différents protocoles tels que IRC (Internet Relay Chat), HTTP (Hyper Text Transfer Protocol) et Email [2].

Le chat de relais Internet (IRC) est basé sur un protocole de messagerie instantanée sur Internet et fonctionne sous une architecture client-serveur. Le Botmaster crée un canal IRC sur le serveur de commande et de contrôle (C&C) et envoie les commandes aux clients [8].

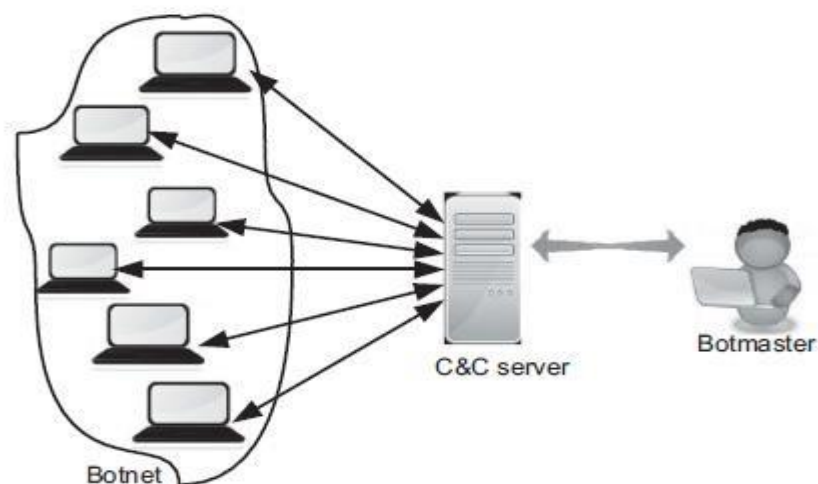


Figure 2.2 : Architecture client-serveur des Botnets [8]

#### 2.4.2 P2P Botnets :

La conception Client-serveur présente un inconvénient majeur d'avoir un seul point de défaillance. Par conséquent, certains attaquants ont utilisé une technologie P2P pour C&C, où chaque bot communique avec un sous-ensemble d'autres bots du réseau, les Botnets P2P sont plus complexes que les Botnets centralisés (Client-serveur).

Dans cette architecture, les bots communiquent entre eux plutôt que via un serveur C&C. Chaque bot conserve une liste de ses voisins. Lorsqu'il reçoit une commande d'un de ses voisins, le bot envoie cette commande aux autres voisins de la liste. Ce scénario se traduit par un réseau appelé **réseau zombie**. Une fois qu'un Botmaster obtient un accès à un hôte du réseau zombie, le Botmaster obtient un contrôle total sur le réseau de Botnet. Chaque hôte du réseau P2P agit à la fois comme client et comme serveur, car il n'y a pas de point centralisé dans cette architecture [8].

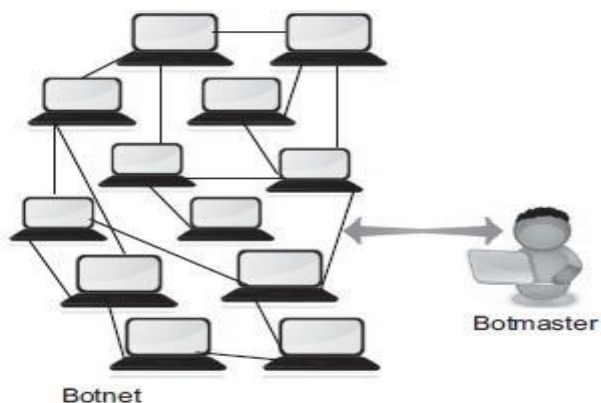


Figure 2.3 : Architecture Botnet P2P [8]

## 2.5 Les différents types d'attaques Botnet :

En général, les Botnets sont considérés comme des sources majeures de différents types d'attaques et d'activités malveillantes sur Internet. Cela comprend les éléments suivants :

**Attaques DDoS** : les Botnets sont utilisés pour lancer plusieurs formes / types d'attaques DDoS. Les bots sont chargés de submerger le système cible avec un volume de trafic élevé (par exemple, des requêtes HTTP, des paquets SYN et des requêtes DNS) [8].

**Usurpation d'identité** : les Botmasters ont la possibilité de collecter des informations sensibles (telles que les comptes de messagerie, les comptes bancaires et les numéros de carte de crédit) à partir des machines de bot [8].

**Crypto-monnaie** : La puissance de calcul des machines qui appartiennent à un Botnet peut être utilisée par les Botmasters pour effectuer une extraction de cryptomonnaie pour obtenir des bitcoins de manière illégale [8].

**Adware** : Les logiciels publicitaires (adware) sont utilisés pour attirer les utilisateurs en faisant de la publicité sur des pages Web ou des applications. Ils apparaissent sur les machines sans l'autorisation des utilisateurs. Les adwares ressemblent à des publicités sans danger mais utilisent des spywares pour collecter les données du navigateur [8].

## 2.6 Les méthodes de détection de Botnet :

Le Botnet est devenu la menace la plus sérieuse pour la cybersécurité. Sa détection devient le sujet de recherche le plus populaire en raison de son grand réseau, de son changement de comportement et de sa robustesse. Les Botmasters développent de nouveaux types de Botnets cryptés qui peuvent cacher leur présence et devenir plus difficiles à retracer. Selon des travaux antérieurs, les techniques de détection de Botnet sont divisées en deux catégories : techniques basées sur Honeynet et les techniques basées sur système de détection d'Intrusion, qui sont en outre divisées en cinq parties : les techniques basées-signature, basées-anomalie, basées-DNS, basées sur les fouilles de données (Data Mining) et basées machine Learning [10].

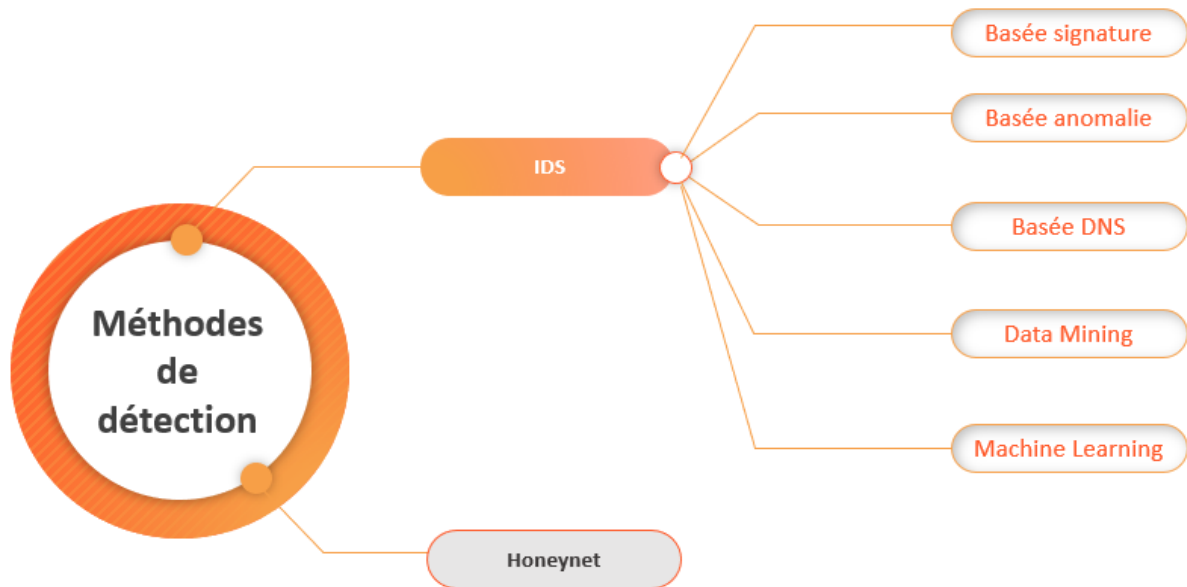


Figure 2.4 : Classification des différentes méthodes de détection des Botnets

### 2.6.1 Détection basée sur les honeynets :

La détection basée sur Honeynet fonctionne avec les Honeybots et un Honeywall, cette technique est utilisée pour détecter et surveiller le comportement du Botnet.

De nombreux chercheurs ont discuté de la détection du Botnet existant à l'aide de Honeynet comme il crée un environnement le mieux adapté pour comprendre le comportement du Bot et ses caractéristiques. Le terme honeypot fait référence à une ressource vulnérable qui peut être facilement compromise. Elle est également vulnérable avec l'intention de devenir une partie du Botnet et attire le Botmaster pour l'infecter. Honeywall est un terme utilisé pour les logiciels utilisés pour collecter et contrôler les informations provenant des Honeybots [10].

### 2.6.2 Détection basée sur les IDS (Système de Détection des Intrusions) :

#### 1) Basé sur la signature (Signature-Based)

Les techniques basées sur la signature fonctionnent à l'aide des signatures des Botnets existants, elles créent une base de données avec des signatures de ces Botnets. Ensuite, en utilisant la méthode de filtrage par motif (Pattern Matching), l'IDS compare la signature du trafic réseau avec celle des bots existants. Grâce à l'existence de signatures dans une base de données, cette technique peut immédiatement trouver le Botnet existant.



Une limitation de cette technique est qu'elle ne peut trouver que les Botnets qui sont connus et déjà tracés [10].

## **2) Basé sur une anomalie (Anomaly-Based)**

Les techniques de détection basées sur les anomalies tentent de détecter les Botnets en fonction de plusieurs anomalies du trafic réseau telles qu'une latence élevée du réseau, des volumes de trafic élevés, du trafic sur des ports inhabituels et un comportement système inhabituel qui pourrait indiquer la présence de bots malveillants sur le réseau [10].

## **3) Basé sur DNS (DNS-Based)**

La technique basée sur le protocole DNS (Domain Name System) est l'une des techniques les plus populaires pour détecter le Botnet. Il s'agit d'une combinaison de techniques basées sur les anomalies et sur les signatures. Les techniques basées sur DNS fonctionnent avec les informations collectées à partir des requêtes DNS. Le Botnet a besoin de chaque bot pour se connecter et communiquer avec le serveur de commande et de contrôle, il doit donc utiliser des requêtes DNS. En surveillant le trafic DNS, l'existence de Botnet peut facilement être détectée. Il peut également tracer l'emplacement du serveur C&C et de Botmaster de Botnet. À l'aide de techniques basées sur DNS, une liste de trous noirs basée sur DNS est créée (**DNS based black-hole list**), qui publie l'emplacement des réseaux ou des ordinateurs à partir desquels des activités malveillantes sont effectuées [10].

## **4) Basé sur la fouille de données (Data Mining-Based)**

Les techniques basées sur les anomalies (Anomaly-Based) sont principalement basées sur des anomalies de comportement du réseau telles qu'une latence élevée du réseau, des activités sur des ports inutilisés...etc. Cependant, le trafic C&C ne révèle généralement pas de comportement anormal. Il est généralement difficile de différencier le trafic C&C du trafic habituel. De ce fait, pour détecter ce type de trafic les techniques de data mining sont utilisées [10].

## **5) Détection par machine Learning (apprentissage automatique)**

Différentes techniques d'apprentissage automatique telles que les classifieurs comme les arbres de décision, les forêts aléatoires etc. sont également utilisées pour détecter les Botnets. Avec les techniques de classification, un grand nombre de trafic peut être

analysé et il devient facile de trouver des bots avec un taux minimisé de faux positif [10].

Technique de détection	Détection de bots connue	Détection de bots inconnue	Taux de faux positif et taux de faux négatif
<b>Basé-anomalie</b>	Oui	Oui	Réduit
<b>Basé-signature</b>	Oui	Non	Elevé
<b>Basé-DNS</b>	Oui	Oui	Elevé
<b>Data mining</b>	Oui	Non	Réduit
<b>Machine Learning</b>	Oui	Oui	Réduit (très réduit en cas de deep Learning)
<b>Honeynet</b>	Oui	Oui	Réduit

Tableau 2.1 : Comparaison des différentes techniques de détection de Botnet [10]

## 2.7 Machine Learning (apprentissage automatique) :

### 2.7.1 Définition :

L'apprentissage automatique est un sous-domaine de l'informatique qui concerne la construction d'algorithmes qui, pour être utiles, s'appuient sur une collection d'exemples de certains phénomènes. Ces exemples peuvent provenir de la nature, être fabriqués à la main par des humains ou générés par un autre algorithme.

L'apprentissage automatique peut également être défini comme le processus de résolution d'un problème pratique par 1) la collecte d'un jeu de données (**dataset**) et 2) la construction d'un modèle statistique basé sur ce jeu de données [11].

### 2.7.2 Types d'apprentissage :

L'apprentissage peut être supervisé, semi-supervisé, non supervisé, renforcé et autres mais les types les plus utilisés sont l'apprentissage supervisé et non-supervisé [11].

**Apprentissage supervisé** : L'apprentissage supervisé est celui où vous pouvez considérer que l'apprentissage est guidé par un enseignant. Nous avons un dataset qui

agit comme un enseignant et son rôle est de former le modèle (**Training**). Une fois que le modèle est bien formé, il peut commencer à faire une prédiction ou une décision lorsque de nouvelles données lui sont fournies [12].

En d'autres termes le but d'un algorithme d'apprentissage supervisé est d'utiliser le dataset pour produire un modèle qui prend un vecteur de caractéristiques (attributs) x (**feature vector**) en entrée et génère des informations qui permettent de déduire l'étiquette (**label**) ou la classe de ce vecteur d'attributs en sortie. Par exemple, le modèle créé à l'aide d'un jeu de données de personnes pourrait prendre en entrée un vecteur de caractéristiques décrivant une personne et produire une probabilité que la personne souffre d'un cancer [11].

**Apprentissage non-supervisé :** Le modèle apprend par l'observation, une fois que le modèle a reçu un dataset, il trouve automatiquement des relations dans le dataset en créant des clusters dans celui-ci. Ce qu'il ne peut pas faire, c'est ajouter des étiquettes au cluster, par exemple il ne peut pas dire cela un groupe de pommes ou de bananes, mais il séparera toutes les pommes des bananes.

L'apprentissage non supervisé est l'apprentissage où vous n'avez que des données d'entrée (X) et aucune variable de sortie correspondante. L'apprentissage est appelé non-supervisé car contrairement à l'apprentissage supervisé ci-dessus il n'y a pas une phase d'apprentissage en utilisant le jeu de données. Les algorithmes sont laissés à eux-mêmes pour découvrir et présenter la structure intéressante des données. Les problèmes d'apprentissage non supervisés peuvent être regroupés en problèmes de clustering et d'association [12].

### 2.7.3 Les modèles de machine Learning :

L'exécution de l'apprentissage automatique implique la création d'un modèle, qui est formé à partir d'un jeu de données d'apprentissage (**training set**) et peut ensuite traiter des données supplémentaires pour faire des prédictions. Différents types de modèles ont été utilisés pour les systèmes d'apprentissage automatique.

Dans l'apprentissage supervisé, il existe deux sous-catégories de modèles : la régression et la classification.

#### **Régression :**

Dans les modèles de régression, l'output ou la sortie est un résultat continu. Voici quelques types de modèles de régression les plus courants.

## Régression linéaire (Linear Regression) :

L'objectif de la régression linéaire est de modéliser la relation entre un ou plusieurs attributs (**features**) et une variable cible continue.

Le but de la régression linéaire simple est de modéliser la relation entre un seul attribut  $x$  et une variable cible  $y$  par une équation linéaire  $y = ax + b$

Nous pouvons également généraliser le modèle de régression linéaire à un modèle avec plusieurs attributs ( $x_1, x_2, \dots, x_n$ ) ; ce processus est appelé régression linéaire multiple avec  $y = \sum_{i=1}^n a_i x_i$  [13].

## Arbre de décision - Régression :

L'arbre de décision construit des modèles sous la forme d'une structure arborescente. Il décompose le jeu de données (dataset) en sous-ensembles de plus en plus petits, tout en développant progressivement un arbre de décision associé. Le résultat final est un arbre avec des nœuds de décision et des feuilles.

Nous développons un arbre de décision en divisant itérativement le jeu de données jusqu'à ce qu'un critère d'arrêt soit satisfait [13].

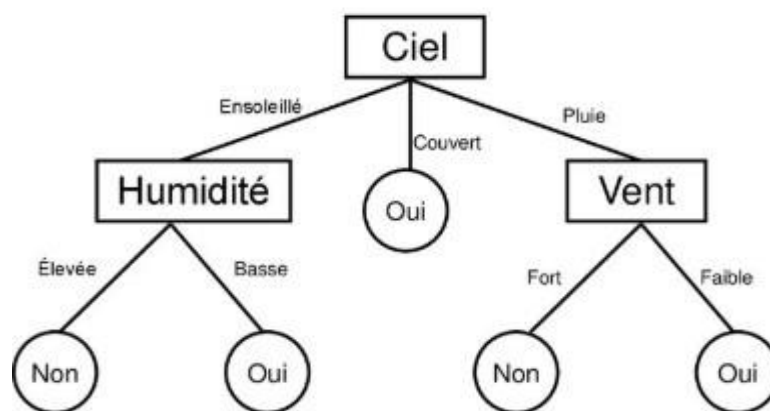


Figure 2.5 : Exemple d'un arbre de décision

## Forêt d'arbres décisionnels aléatoire (Random Forest) - Régression :

L'algorithme de forêt aléatoire est une technique qui combine plusieurs arbres de décision, une forêt aléatoire a généralement une meilleure performance qu'un arbre de décision individuel en raison du caractère aléatoire qui réduit le taux d'erreur.

L'algorithme des « forêts aléatoires » effectue un apprentissage en parallèle sur de multiples arbres de décision construits aléatoirement et formés sur des sous-ensembles de données différentes. Le nombre idéal d'arbres, qui peut aller jusqu'à plusieurs

centaines voire plus, est un paramètre important (il est très variable et dépend du problème). Concrètement, chaque arbre de la forêt aléatoire est formé sur un sous-ensemble aléatoire de données, avec un sous ensemble aléatoire des attributs (caractéristiques des données). Des prédictions de chaque arbre pour des nouvelles données sont ensuite utilisées pour un vote et la classe avec le plus de votes devient la prédiction de notre modèle en cas de classification ou une moyenne est calculée à partir des prédictions résultantes en cas de régression [13].

### **Classification :**

Dans les modèles de classification, l'output ou la sortie est un résultat discret. Voici quelques types de modèles de classification les plus courants.

### **Régression logistique (Logistic Regression) :**

La régression logistique est nommée pour la fonction utilisée au cœur de la méthode, la fonction logistique également appelée fonction sigmoïde.

La régression logistique utilise une équation tout comme la régression linéaire. Les valeurs d'entrée (x) sont combinées linéairement en utilisant des poids ou des valeurs de coefficient pour prédire une valeur de sortie (y). Une différence essentielle par rapport à la régression linéaire est que la valeur de sortie modélisée est une valeur binaire (0 ou 1) plutôt qu'une valeur numérique [12].

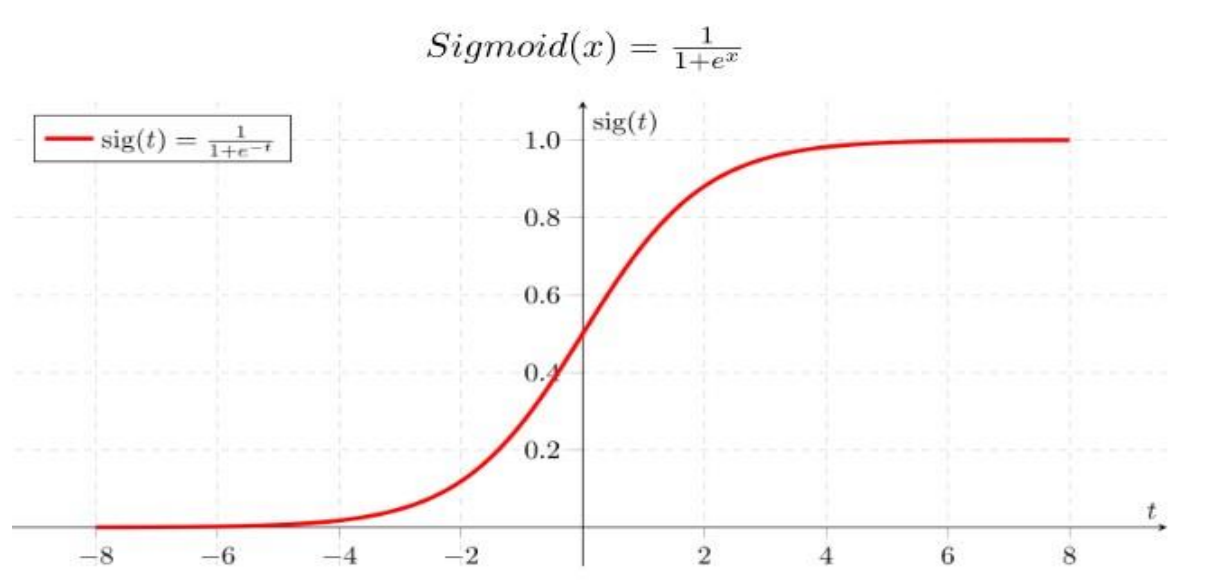


Figure 2.6 : La courbe qui définit la fonction logistique (fonction sigmoïde) [15]

Voici quelques exemples de problèmes de classification logistique binaire :

- Détection de spam : prédire si un e-mail est du spam ou non.
- Fraude par carte de crédit : prédire si une transaction par carte de crédit donnée est une fraude ou non.
- Santé : prédire si une masse donnée de tissus est bénigne ou maligne.

### **Machine à vecteurs de support (Support Vector Machine) :**

Une machine à vecteur de support est une technique de classification supervisée qui peut être considérée comme une extension du perceptron (**Le perceptron** est un algorithme d'apprentissage supervisé de classification binaire « c'est-à-dire séparant deux classes [16] » où des détails sur ce type d'apprentissage seront abordés dans la partie d'apprentissage profond). En utilisant l'algorithme perceptron, les erreurs de classification erronée sont minimisées. Cependant, dans les SVM, l'objectif est de maximiser la marge. La marge est définie comme la distance entre l'hyperplan de séparation (une limite de décision) et les échantillons d'apprentissage les plus proches de cet hyperplan, qui sont les soi-disant vecteurs de support [13].

Les variables en entrée (x qui sont les attributs) forment un espace à n dimensions. Par exemple, si vous aviez deux variables d'entrée, cela formerait un espace à deux dimensions. Dans SVM, un hyperplan est sélectionné pour séparer au mieux les points dans l'espace variable d'entrée selon leur classe, soit la classe 0 ou la classe 1 [12].

### **Naïve Bayes :**

En apprentissage automatique, nous sommes souvent intéressés à sélectionner la meilleure hypothèse (h) sachant une donnée (d).

Dans un problème de classification, notre hypothèse (h) peut être la classe à attribuer à une nouvelle instance de données (d). L'une des façons les plus simples de sélectionner l'hypothèse la plus probable sachant une donnée est d'utiliser le théorème de Bayes. Le théorème de Bayes est formulé comme suit [12] :

$$P(h|d) = \frac{P(d|h) \times P(h)}{P(d)}$$

Où :

- $P(h|d)$  est la probabilité de l'hypothèse h sachant les données d. C'est ce qu'on appelle la probabilité postérieure.

- $P(d|h)$  est la probabilité des données  $d$  sachant que l'hypothèse  $h$  était vraie.
- $P(h)$  est la probabilité que l'hypothèse  $h$  soit vraie (quelles que soient les données  $d$ ).
- $P(d)$  est la probabilité des données (quelle que soit l'hypothèse  $h$ ).

### **Les K plus proches voisins (KNN : K-Nearest Neighbours) :**

KNN est un algorithme d'apprentissage paresseux. Il est appelé paresseux pas en raison de sa simplicité apparente, mais parce qu'il n'apprend pas une fonction discriminante à partir des données d'apprentissage, mais mémorise à la place le jeu de données d'apprentissage [13].

L'algorithme KNN lui-même est assez simple et peut être résumé par les étapes suivantes [13] :

- Choisissez le nombre de  $k$  et une distance de l'échantillon (ou l'observation) par rapport aux voisins les plus proches.
- Trouvez les  $k$  voisins les plus proches de l'échantillon que nous voulons classer.
- Attribuez la classe par vote majoritaire.

KNN effectue des prédictions en utilisant directement le jeu de données d'apprentissage. Des prédictions sont faites pour un nouveau point de données en recherchant dans le jeu de données d'apprentissage les  $K$  instances les plus similaires (les voisins) et une classe est attribuée selon le nombre majoritaire des voisins dans une des classes [12].

### **Arbre de décision, forêt d'arbres décisionnels aléatoire -classification :**

Ces modèles suivent la même logique que celles de régression, la seule différence est que l'output est un résultat discret plutôt que continu.

Une des méthodes la plus souvent utilisée est le clustering dans l'apprentissage non supervisé.

### **Clustering :**

Un problème de clustering est le problème où vous souhaitez découvrir les regroupements des données, comme le regroupement des clients par comportement d'achat par exemple ou classification du document.

Les techniques de clustering incluent le clustering k-means, le clustering hiérarchique, le clustering à décalage moyen et le clustering basé sur la densité. Bien que chaque technique ait une méthode différente pour trouver les regroupements, elles visent toutes à atteindre la même chose [12].

## 2.8 Deep Learning (l'apprentissage profond) et réseau de neurones :

### 2.8.1 Définition de l'apprentissage profond :

Nous pourrions définir l'apprentissage profond comme une classe de techniques d'apprentissage automatique, où les informations sont traitées en couches hiérarchiques où chaque couche va prendre en entrée les résultats ou la sortie de la couche précédente.

L'apprentissage profond peut être compris comme un ensemble d'algorithmes qui ont été développés pour former les réseaux de neurones **profonds** (**DNN : Deep Neural Networks**) de manière plus efficace.

En pratique, tous les algorithmes d'apprentissage en profondeur sont appliqués sur les réseaux de neurones profonds qui sont des réseaux de neurones avec multiples couches. Les réseaux de neurones partagent certaines propriétés de base communes. Ils sont tous constitués de neurones interconnectés organisés en couches. Ils peuvent différer dans l'architecture du réseau (ou la façon dont les neurones sont organisés dans le réseau), et parfois la manière dont ils sont entraînés. Dans cet esprit, regardons les principaux types de réseaux de neurones [20].

Il existe plusieurs types de réseaux de neurones, nous citons les plus utilisées :

- Perceptron (une couche d'entrée et une de la sortie)
- Perceptron multicouche (**MLP : Multi-layer perceptron**).
- Réseau de neurones convolutif (**CNN : Convolutional Neural Network**).
- Réseau de neurones récurrent (**RNN : Recurrent Neural Network**).
- Les encodeurs automatiques (**Auto-encoder**)

Le premier est considéré comme un réseau de neurones simple mais les autres types sont considérés comme des réseaux de neurones profonds comme ces réseaux comportent une couche d'entrée et une couche de sortie et des couches supplémentaires.



## 2.8.2 Domaines d'application d'apprentissage profond :

L'apprentissage profond est utilisé dans de nombreux domaines :

- Reconnaissance d'image,
- Traduction automatique,
- Diagnostic médical,
- Recommandations personnalisées,
- Détection de malwares ou de fraudes,
- Chatbots (agents conversationnels),
- Robots intelligents.

## 2.8.3 Réseaux de neurones artificiels :

Le concept de base derrière les réseaux de neurones a été construit sur des hypothèses et des modèles de la façon dont fonctionne le cerveau humain pour résoudre des tâches complexes.

Les réseaux de neurones simulent le mécanisme d'apprentissage dans le système nerveux. Le système nerveux humain contient des cellules, appelées neurones. Les neurones sont connectés les uns aux autres à l'aide d'axones et de dendrites, et les régions de connexion entre les axones et les dendrites sont appelées synapses.

Ce mécanisme biologique est simulé dans des réseaux de neurones artificiels, qui contiennent des unités de calcul appelées neurones. Les unités de calcul sont connectées les unes aux autres par des poids, qui jouent le même rôle que les forces des connexions synaptiques dans le système nerveux. Chaque entrée d'un neurone est mise à l'échelle avec un poids (le poids représente l'importance de ce neurone dans la prédiction de résultat ce qui affecte la fonction calculée à cette unité).

Un réseau de neurones artificiels calcule une fonction en propageant les valeurs calculées des neurones d'entrée aux neurones de sortie et en utilisant les poids comme paramètres intermédiaires. L'apprentissage se fait en modifiant les poids reliant les neurones. Tout comme des stimuli externes sont nécessaires pour l'apprentissage dans le système nerveux, le stimulus externe dans les réseaux de neurones artificiels est fourni par les données d'apprentissage (**Training data**) contenant des exemples de paires entrée-sortie de la fonction à apprendre. Les données d'apprentissage fournissent un aperçu sur l'exactitude des poids dans le réseau de neurones en fonction de l'adéquation de la sortie prévue pour une entrée particulière.

Les poids entre les neurones sont ajustés dans un réseau de neurones par une fonction appelé rétro-propagation (**Backpropagation**) en réponse à des erreurs de prédiction. Le but de changer les poids est de modifier la fonction calculée pour rendre les prédictions plus correctes dans les futures itérations. Par conséquent, les poids sont soigneusement modifiés afin de réduire l'erreur de calcul. En ajustant successivement les poids entre les neurones sur de nombreuses paires entrée-sortie, la fonction calculée par le réseau de neurones est affinée au fil du temps afin de fournir des prédictions plus précises [21].

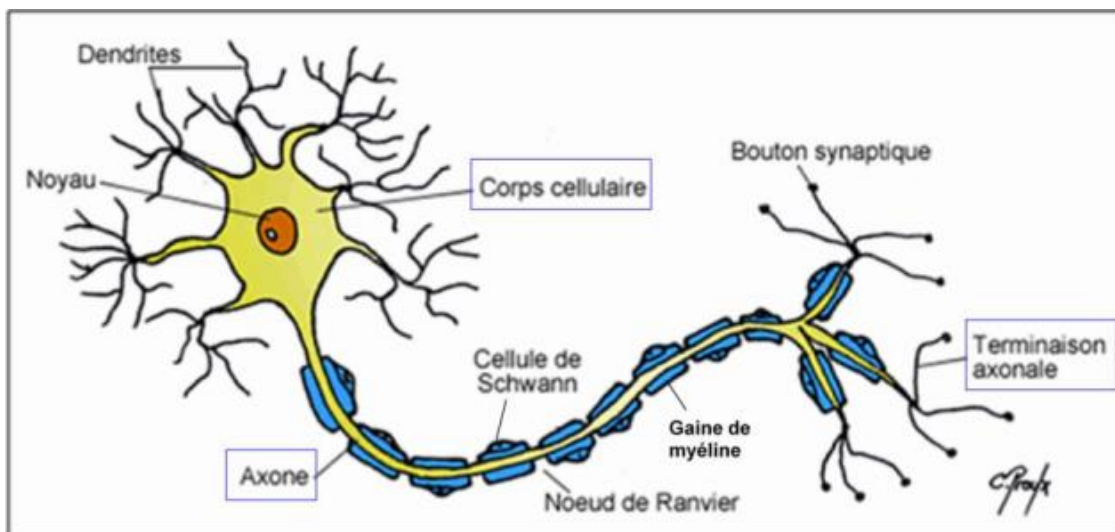


Figure 2.7 : Structure générale d'un neurone [22]

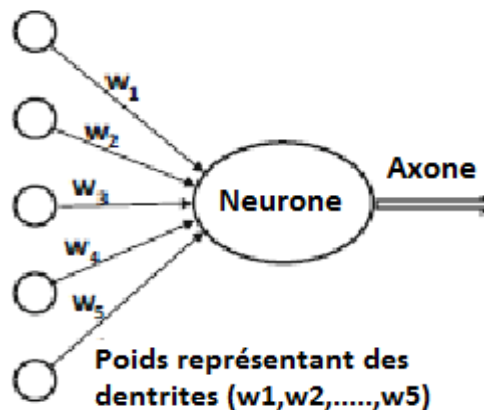


Figure 2.8 : la représentation de système nerveux dans le réseau de neurones artificiel [21]

### 2.8.4 Réseau de neurones monocouche (Le perceptron) :

Le réseau de neurones le plus simple est appelé le perceptron. Ce réseau contient une seule couche d'entrée et un nœud de sortie.

La couche d'entrée contient des nœuds qui transmettent les  $d$  caractéristiques (attributs)  $X = [x_1 \dots x_d]$  avec des poids correspondants  $W = [w_1 \dots w_d]$  vers le nœud de sortie. La couche d'entrée n'effectue aucun calcul, la fonction linéaire  $W \cdot X = \sum_{i=0}^d w_i x_i$  est calculée au nœud de sortie. Par la suite, le résultat obtenu de la fonction linéaire sera l'entrée pour une fonction d'activation utilisée afin de prédire la classe pour le vecteur des caractéristiques  $X$  [21].

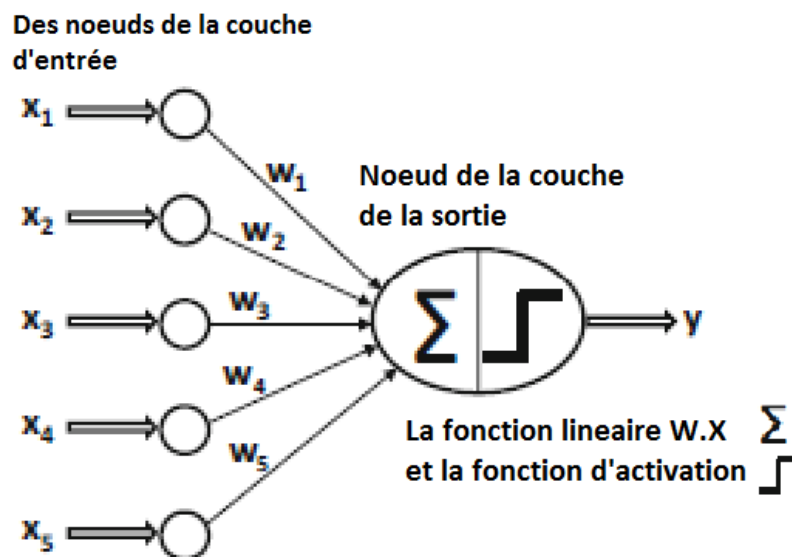


Figure 2.9 : Réseau de neurones monocouche [21]

### 2.8.5 Réseau de neurones multicouche :

Les réseaux de neurones multicouches contiennent plusieurs couches, comme le perceptron les réseaux de neurones multicouches contiennent une couche d'entrée et autre de sortie au plus de ça ils contiennent des couches intermédiaires supplémentaires entre ces deux qui sont appelées couches cachées (**Hidden layers**). Les couches successives s'alimentent dans le sens de l'entrée à la sortie (les résultats des calculs des nœuds de la couche précédente sont des entrées pour la fonction linéaire des nœuds de la couche suivante « **feed-forward neural network** »). L'architecture des réseaux multicouche suppose que tous les nœuds d'une couche sont connectés à ceux de la couche suivante [21].

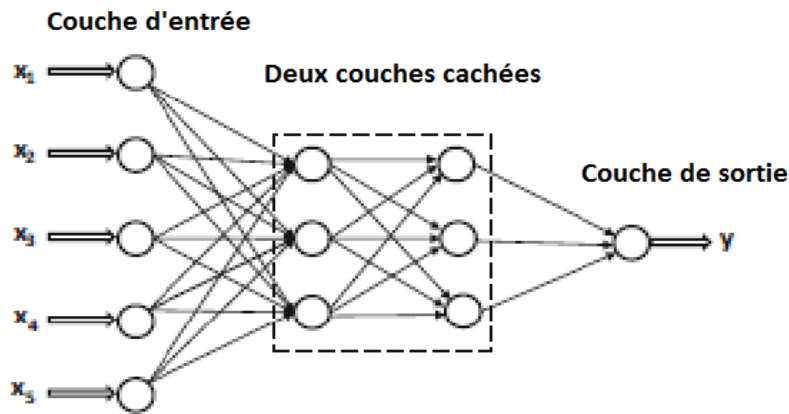


Figure 2.10 : réseau de neurones multicouche avec deux couches cachées [21]

### 2.8.6 Les différents types de fonction d'activation :

Habituellement, tous les neurones d'une même couche ont la même fonction d'activation, mais différentes couches peuvent avoir des fonctions d'activation différentes. Les fonctions d'activation les plus courantes sont les suivantes : (Dans ce tableau les valeurs de  $x$  sont les valeurs de la fonction linéaire  $W \cdot X = \sum_{i=0}^d w_i x_i$  où  $X$  contient les valeurs des différents attributs et  $W$  sont les poids correspondants) [20]

Nom	Graph	Fonction
Fonction de signe		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$
Fonction logistique (sigmoïde)		$f(x) = \frac{1}{1 + e^{-x}}$
Fonction Unité de rectification linéaire (ReLU)		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$
Tangente hyperbolique		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$

Tableau 2.2 : Les différents types de fonction d'activation [20]

### 2.8.7 Fonctions de propagation et rétropropagation :

Les réseaux de neurones sont généralement formés selon deux phases une phase de propagation et une autre de rétropropagation. Une fois qu'une observation (ou plus souvent un nombre défini d'observations appelé taille du lot (**Batch Size**)) est acheminée via le réseau, la valeur résultante est comparée à la valeur réelle de l'observation à l'aide d'une fonction de perte (**Loss Function**) celui ce qu'on appelle la propagation. Ensuite, un algorithme de rétropropagation va s'exécuter en sens inverse à travers le réseau en identifiant dans quelle mesure chaque paramètre qui a contribué à l'erreur entre les valeurs prévues et vraies. À chaque paramètre, l'algorithme d'optimisation détermine combien de poids doit être ajusté pour améliorer les résultats de sortie.

Les réseaux de neurones apprennent en répétant ce processus de propagation vers l'avant et de rétropropagation pour chaque observation dans les données d'apprentissage (Training set) plusieurs fois en mettant à jour de manière itérative les valeurs des poids (chaque fois que toutes les observations ont été envoyées via le réseau est appelé **une époque** et l'apprentissage se compose généralement de plusieurs époques et chaque époque contient plusieurs **itérations** où chaque itération est le processus de propagation et rétropropagation pour un lot (**batch**) d'observations selon sa taille ) [23].

#### Quelques fonctions de perte :

##### **Fonction d'erreur quadratique (MSE : Mean Square Error) :**

L'équation ci-dessous compare la sortie réelle ( $y$ ) du réseau de neurones avec la sortie prédite ( $\hat{y}$ ). La variable  $n$  contient le nombre d'éléments d'apprentissage multiplié par le nombre de neurones de sortie [24].

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

##### **Fonction d'erreur d'entropie croisée (CE Cross-Entropy) :**

$$CE = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln (1 - a)]$$

Avec :

- $y$  - indicateur binaire (0 ou 1) pour indiquer si la classification est correcte pour une observation de training set.
- $a$  - probabilité qu'une observation appartienne à la classe prédite [24].

## 2.9 Comparaison d'apprentissage profond et l'apprentissage automatique :

Facteurs	Machine Learning	Deep Learning
Volume de données	Jeu de données de taille petite ou moyenne	Big data : Des larges jeux de données
Dépendance matérielle	N'exige pas une machine puissante	Nécessite une grande puissance de traitement et du matériel coûteux
Temps d'exécution	Des minutes vers des heures	Des Heures
Précision	Acceptable	Excellente

Tableau 2.3 : comparaison entre l'apprentissage automatique et l'apprentissage profond

## 2.10 Travaux antérieurs :

Les techniques de détection de Botnet ont commencé avec des techniques basées sur des signatures et évoluent vers des approches plus sophistiquées. Les approches récentes de la littérature sur les Botnets ciblent les caractéristiques de flux réseau et appliquant des algorithmes d'apprentissage automatique ou profond pour les classifications des Botnets. Des algorithmes supervisés et non supervisés sont utilisés dans ces approches.

Livadas et al. [17] a introduit pour la première fois l'utilisation d'algorithmes d'apprentissage automatique dans les techniques de détection de Botnets. Ce travail dépend du protocole et se concentre uniquement sur le trafic de protocole IRC pour détecter le Botnet. Des algorithmes l'arbre de décision J48 et Naïve Bayes ont été appliqué pour classifier le trafic IRC en botnet ou normal.

Les travaux proposés par Beigi et al. [18] utilisent un jeu de données (dataset) du monde réel créé par la combinaison de 16 traces botnets connus et introduisent une grande diversité en incluant plus de 40% d'échantillons de Botnet inconnus dans les données de test. L'approche applique un algorithme supervisé basé sur l'arbre de décision C4.5 pour la détection des Botnets et un algorithme gourmand est appliqué pour la sélection des caractéristiques (**feature selection**). Le travail montre une exactitude de détection de 75%.

Le travail proposé par Farhan et al. [19] a exploré le protocole OpenFlow pour la détection de Botnet. L'approche collecte de manière centralisée les statistiques de flux réseau OpenFlow à partir du contrôleur SDN. Cette méthode utilise également un jeu de données des Botnets disponibles dans le monde réel CTU-13. Le travail injecte des échantillons de Botnets inconnus dans des données de test. L'algorithme de classification supervisée basé sur l'arbre de décision C4.5 est utilisé et les expériences montrent un taux de détection de 80%.

Le travail [25] de Tang et al. proposent une approche de deep Learning pour la détection d'intrusions en général dans les réseaux SDN en utilisant un STL (selftaught Learning) et le dataset NSL-KDD. En particulier, ils exploitent un réseau de neurones à cinq couches avec une dimension de six neurones par couche et une dimension de deux neurones pour la couche de sortie, le travail montre une exactitude de 75.75% et une précision de 83%.

Le travail de Aboubakar et al. [26] présente un système de détection d'intrusion basée sur l'apprentissage profond (réseau de neurones) pour le SDN en utilisant le dataset NSL-KDD. Le modèle IDS est basé sur les flux pour détecter les attaques basées sur des anomalies dans l'environnement SDN, le modèle a atteint une exactitude de 97.3%

Cependant, pour ces travaux, le problème abordé est beaucoup plus général (pas spécifique au Botnet) comme le travail [25] et [26], pas spécifique au SDN comme dans [17] et [18], les auteurs exploitent des jeux de données obsolètes le cas de travail dans [25] et [26] ou les performances ne sont pas satisfaisantes comme les travaux [18] et [25] où l'exactitude n'a pas dépassé 75%. Pour le travail [19] l'exactitude était bonne mais le taux des faux négatifs était un peu élevé jusqu'à 20.44% comme les attributs sélectionnés sont insuffisants pour une excellente exactitude et un petit taux de faux négatif.

## 2.11 Conclusion :

Au cours de ce chapitre, nous avons tout d'abord étudié le problème de Botnet dans les réseaux SDN. Nous avons ensuite présenté quelques méthodes et modèles de détection des attaques Botnet et enfin nous avons présenté les algorithmes de l'apprentissage automatique et ceux de l'apprentissage profond et quelques travaux antérieurs. Dans le prochain chapitre, nous allons proposer une méthode pour détecter les attaques Botnet en basant sur les algorithmes de l'apprentissage profond.



# Chapitre 3 : Approche proposée pour la détection des Botnets dans le SDN

## 3.1 Introduction :

Dans les chapitres précédents, nous avons présenté la technologie SDN et les différentes caractéristiques de cette nouvelle tendance, ainsi que quelques notions sur les Botnets et l'apprentissage automatique et enfin nous avons cité quelques travaux qui ont traité la détection des malwares dans le réseau SDN.

Dans ce chapitre, nous allons présenter notre approche pour la conception d'un système qui permettra de détecter les attaques Botnets. Nous commencerons par définir l'aspect architectural de notre solution, nous enchaînerons ensuite avec l'introduction des différents modules de notre système de détection de Botnet.

## 3.2 Architecture générale de notre système :

Pour avoir un système de détection de Botnet performant, l'approche que nous avons proposée a pour but de construire un système à trois modules :

- Un module de collection de flux
- Un module d'extraction des caractéristiques
- Un module d'apprentissage approfondi pour la classification de flux

Chaque module est implémenté dans la couche d'application SDN, comme présenté dans la figure 3.1, et a un rôle qui va être décrit prochainement.

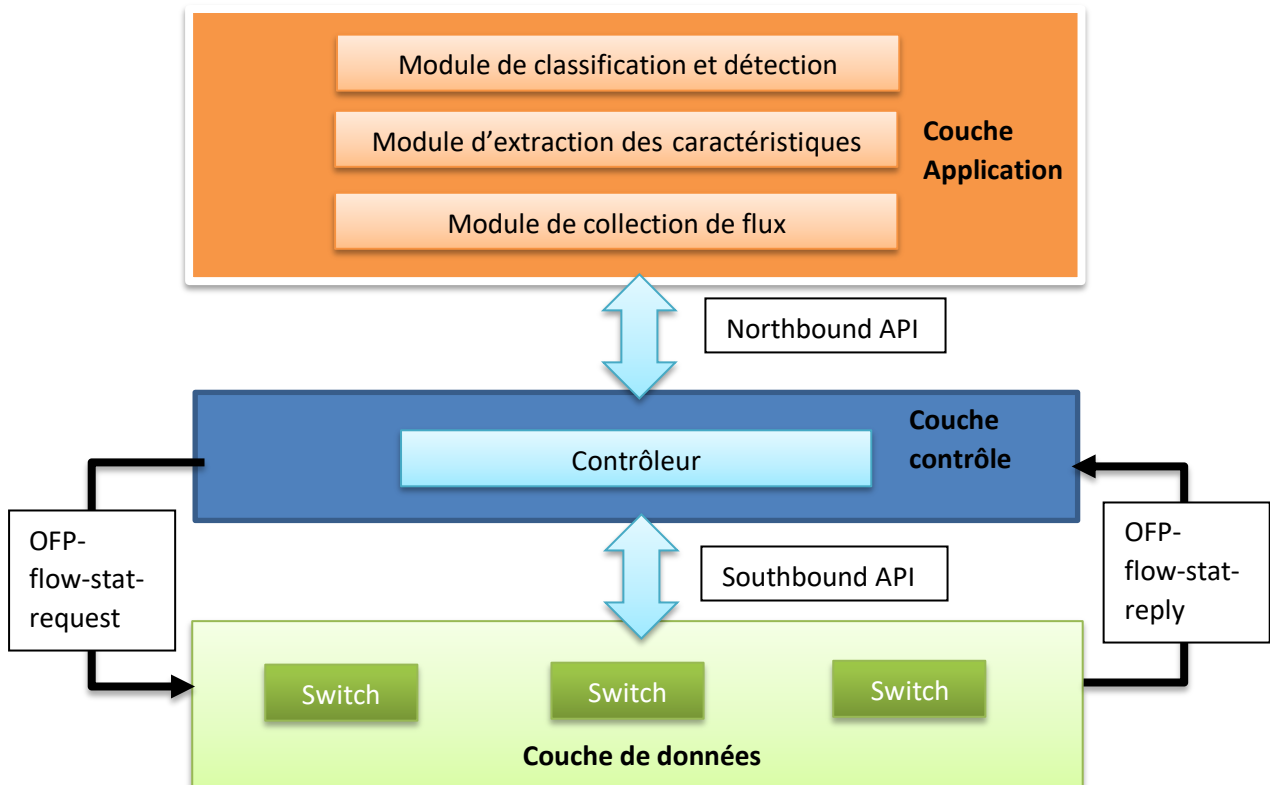


Figure 3.1 : Présentation de l'architecture de système de détection des Botnets

### 3.2.1 Collecteur de flux :

Le fonctionnement de ce module est le suivant :

- Après chaque intervalle de temps le contrôleur envoie une requête afin de collecter les statistiques des différents flux installés dans les tables de flux des différents commutateurs ; cela peut être réalisé grâce au message de protocole OpenFlow **OFP-flow-stat-request**.
- Une fois la requête reçue par le commutateur, celui-ci répond par un message OpenFlow de réponse **OFP-flow-stat-reply**. Ce message contient les différentes statistiques des flux installés dans la table de flux de ce commutateur.

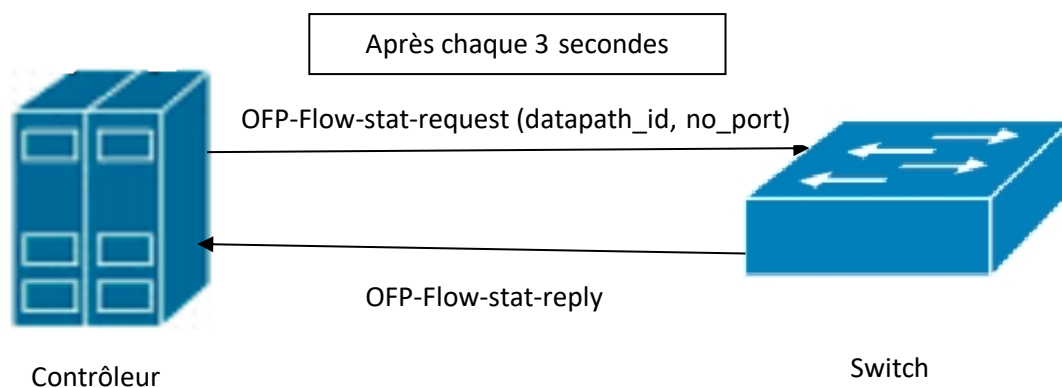


Figure 3.2 : Fonctionnement de collecteur du flux

Le message **OFP-flow-stat-request** est envoyé après chaque 3 secondes à un commutateur spécifique selon les paramètres (datapath\_id) qui est l'identifiant de commutateur et (no\_port) qui est le port utilisé pour envoyer ce message.

Le datapath\_id s'agit d'un nombre de 64 bits déterminé comme suit selon la spécification OpenFlow : les 48 derniers bits sont destinés à l'adresse MAC du commutateur, tandis que les 16 premiers bits sont spécifiés par le fournisseur. Un exemple d'utilisation des 16 premiers bits serait l'ID de VLAN pour distinguer plusieurs instances de commutateurs virtuels sur un seul commutateur physique.

La réponse **OFP-flow-stat-reply** contient les différentes informations et statistiques concernant les flux installés dans la table de flux des différents commutateurs, ces statistiques sont les suivantes :

Information/statistique	Description
datapath_id	L'identifiant unique de switch
Protocol	Type de protocole (ICMP, UDP, TCP)
Packet count	Le numéro de paquets dans le sens bidirectionnel (de source vers destination et de destination vers source)
Bytes count	Le numéro d'octets dans le sens bidirectionnel
Hard time out	Durée d'expiration maximale d'un flux
Idle time out	Temps de permanence du flux (s'il n'y a pas de connections entre la source et destination pendant une période spécifique, le flux va être retiré de la table des flux de switch)

Ip_src	Adresse ipv4 de la source
Ip_dst	Adresse ipv4 de la destination
Udp_src	Numéro de port de source pour protocole UDP
Udp_dst	Numéro de port de destination pour protocole UDP
Tcp_src	Numéro de port de source pour le protocole TCP
Tcp_dst	Numéro de port de destination pour le protocole TCP
Duration	Durée de flux à partir de sa création dans la table des flux

Tableau 3.1: Tableau des différentes statistiques que le module de collection de flux peut extraire à partir de la table de flux d'un switch OpenFlow

### 3.2.2 Module d'extraction des caractéristiques :

Lorsque le contrôleur collecte périodiquement les statistiques des différents flux dont il a besoin, le module d'extraction récupère les caractéristiques à partir de la collection du module précédent et calcule d'autres caractéristiques qui seront nécessaires pour la détection des attaques Botnet.

Une fois l'extraction et les calculs terminés, le module prépare un **dataset** (jeu de données) contenant des vecteurs de caractéristiques représentant les différents flux.

Chaque vecteur de caractéristiques contient les valeurs numériques des différents attributs choisis qui représentent le flux (nombre d'octets, nombre de paquets, la durée de flux, le protocole, temps d'utilisation de flux.....etc.)

La structure du vecteur de caractéristiques peut être considérée comme suit :

Nombre de paquets source du flux	Nombre de paquets destination du flux	Duration du flux	Protocole	Temps d'utilisation du flux	Différence de temps d'arrivée des paquets	..... (Les valeurs des autres attributs)	Nombre total d'octets
----------------------------------	---------------------------------------	------------------	-----------	-----------------------------	---	---	-----------------------

L'algorithme utilisé pour extraire les caractéristiques de chaque flux est le suivant :

*Début*

*Flux = {} /\*Initialiser un dictionnaire des flux qui a pour clé l'identifiant d'un flux et comme valeur le vecteur de caractéristiques pour ce flux\*/.*

*Si les statistiques du collecteur de flux contenant les informations des différents flux sont reçus :*

*unique\_id = adresse\_IP\_src + port\_src + adresse\_IP\_dst + port\_dst /\*Créer un identifiant unique contenant les informations pour chaque flux reçu (Adresse IP source, numéro de port source, Adresse IP destination, numéro de port destination)\*/.*

*Si unique\_id ∈ Flux /\*Vérifier si l'identifiant existe déjà dans le dictionnaire \*/ :*

*mise\_à\_jour\_flux (nbr\_paquets, nbr\_octets, duration\_flux, temps\_actuel....etc.) /\*Mettre à jour le vecteur de caractéristiques de ce flux \*/*

*Sinon :*

*Flux = Flux + {unique\_id, vecteur\_caractéristiques}*

*/\*Création d'une nouvelle entrée dans le dictionnaire. \*/*

*Flux[unique\_id] = [nbr\_paquets, nbr\_octets, duration\_flux, temps\_actuel...etc.]*

*/\*Remplir le vecteur de caractéristiques avec les informations des attributs choisis d'un jeu de données utilisé HogZilla, quelques-unes peuvent être directement extraites à partir des statistiques reçus alors que d'autres doivent être calculées \*/.*

*FinSi*

*Un fichier contenant un jeu de données est créé en regroupant ces vecteurs de caractéristiques.*

*FinSi*

*Fin*

Le jeu de données est utilisé par le dernier module qui va classifier le type de trafic en un trafic normal ou Botnet.

Les différentes caractéristiques choisies **qui seront essentielles** pour la détection d'attaque Botnet et celles qui seront calculées vont être discutées dans le prochain chapitre.

### 3.2.3 Module de classification des flux :

#### **Phase d'apprentissage :**

La phase d'apprentissage se résume en la création et l'entraînement d'un modèle basé DNN qui signifie un modèle basé sur un réseau de neurones artificiel profond.

Les réseaux de neurones approfondis ont été largement utilisés dans les tâches de classification, car ces derniers ont montré une prédiction plus améliorée que celle des algorithmes de machine Learning comme SVM, KNN ou Forêt d'arbres décisionnels aléatoire. Le réseau de neurones est organisé en couches, qui peuvent apprendre et prendre des décisions intelligentes ultérieurement. Alors que dans l'apprentissage automatique, les décisions sont prises uniquement en fonction de ce qui a été appris.

Dans notre travail, nous avons utilisé un réseau de neurones multicouche (**MLP**) avec les couches suivantes :

- Une couche d'entrée
- 3 couches cachées
- Une couche de sortie

#### **Phase de classification :**

En se basant sur le modèle formé (Trained Model) créé en phase d'apprentissage, les vecteurs de caractéristiques du jeu de données construit dans le deuxième module vont être analysées et les flux vont être classifiés en flux normal ou Botnet.

La figure suivante nous montre le fonctionnement de ce module :

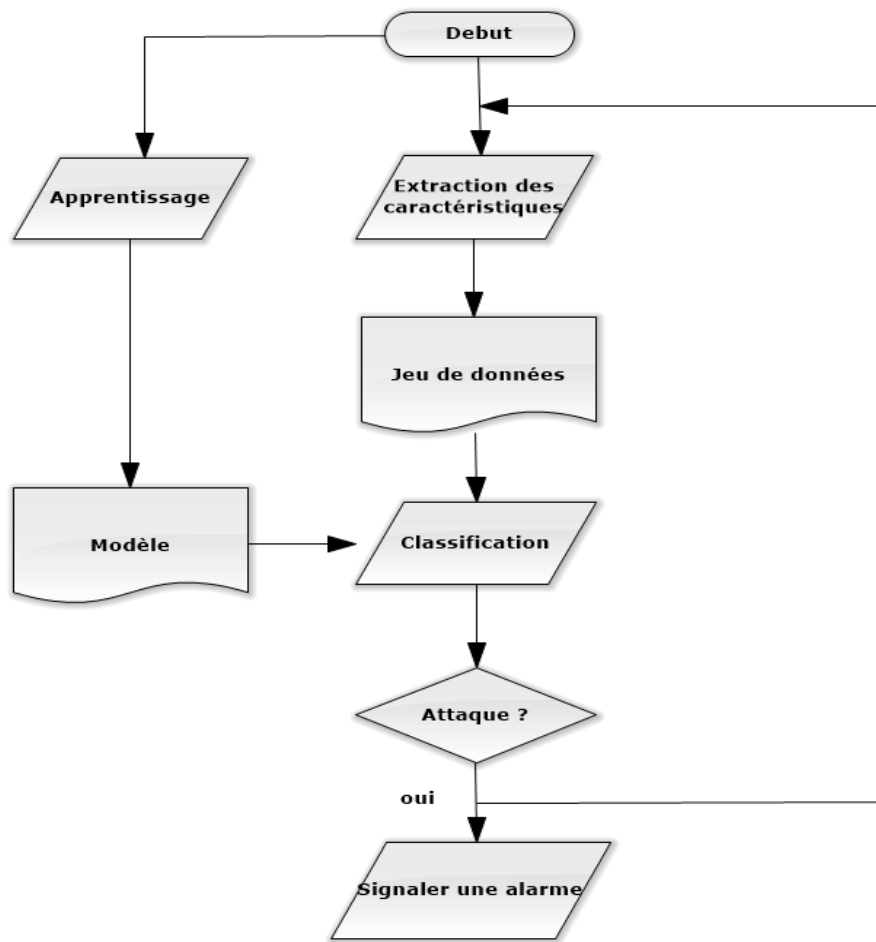


Figure 3.3 : Processus d'apprentissage et classification des flux

### 3.3 Conclusion :

Au cours de ce chapitre, nous avons présenté l'aspect architectural ainsi que l'aspect fonctionnel de notre solution qui permettra la détection des attaques botnets dans la technologie SDN. Nous tenterons ensuite de valider cette solution sur le dataset Hogzilla dans le prochain chapitre.

# Chapitre 4 : Validation sur HogZilla dataset

## 4.1 Introduction :

Après avoir détaillé l'aspect architectural et fonctionnel de notre approche pour la détection des attaques Botnet dans les réseaux SDN, nous présentons dans ce chapitre le jeu de données utilisé pour la phase d'apprentissage de notre réseau de neurones profond.

Dans ce chapitre, nous aborderons le dataset Hogzilla indispensable à la validation de notre solution puis nous citerons les attributs choisis pour la phase d'apprentissage ainsi que la description de ces attributs. Finalement, nous montrerons les différents logiciels, langage de programmation choisis ainsi que les bibliothèques utilisées suivi par les résultats obtenus après la validation de l'approche proposée sur ce jeu de données.

## 4.2 Le jeu de données Hogzilla (Hogzilla dataset) :

Le jeu de données HogZilla est un jeu de données récemment publié obtenu en fusionnant un fragment du jeu de données bien connu CTU-13, qui contient 13 traces réseau de 7 malwares de Botnet distincts, et un grand ensemble de traces de trafic normal extrait de jeu de données IDS ISCX 2012. Ce trafic est encore prétraité et classé, ce qui donne 990 000 échantillons, chacun associé à 192 caractéristiques (attributs) [27].

Pour des résultats adéquates, nous avons utilisé le jeu de données Hogzilla modifié par Ivan Lettri, Giuseppe Della Penna et Giovanni De Gasperis [28] en extrayant un jeu de données équitable et équilibré (**Fair dataset**), c'est-à-dire un jeu de données contenant exactement le même nombre d'échantillons normaux et de Botnet. Ce jeu de données est utilisé pour la phase d'apprentissage du réseau de neurones, car le jeu de données complet présente beaucoup plus de trafic normal que de trafic malveillant, ce qui pourrait biaiser l'apprentissage du réseau de neurones.



### 4.3 La sélection des attributs (Feature Selection) :

La sélection des attributs vise à concevoir le sous-ensemble minimal des attributs suffisant pour détecter le comportement d'un Botnet. Ceci est particulièrement utile pour l'analyse basée sur l'apprentissage automatique, où trop d'attributs à prendre en compte (qui n'ont pas un grand impact sur la classification) peuvent confondre le classifieur.

Nous nous sommes concentrés sur 16 attributs (caractéristiques), en tenant compte de ceux qui conviennent le mieux à la détection de Botnet, celles qui ont le meilleur score selon une méthode de sélection appelée « **La sélection univariée** » et qui peuvent être réellement collectés dans un réseau SDN. En effet, dans un réseau SDN, seules les statistiques comme le nombre de paquets, le nombre d'octets et la durée du flux peuvent être lues à partir du contrôleur (du plan de contrôle), qui l'obtient via les requêtes OpenFlow au plan de données (les messages **OFP\_Flow\_stats\_request** et **OFP\_Flow\_stats\_reply**). Cependant, d'autres attributs utiles peuvent être calculés à partir des statistiques collectés par le contrôleur.

**La sélection univariée :** La sélection univariée des attributs fonctionne en sélectionnant les meilleurs attributs basés sur des scores résultants du calcul d'une formule statistique « **test du  $\chi^2$  ou chi-square** ». Elle examine chaque attribut individuellement pour déterminer la force de la relation de l'attribut avec la variable de sortie [23].

La formule chi-square peut être donné comme suit :

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i}$$

Où  $O_i$  est le nombre d'observations dans la classe  $i$  et  $E_i$  est le nombre d'observations dans la classe  $i$  auquel on s'attendrait s'il n'y a pas de relation entre l'attribut et la variable de sortie. En calculant chi-square entre un attribut et la variable de sortie, nous obtenons une mesure de l'indépendance entre les deux. Si la variable de la sortie est indépendante de l'attribut, l'attribut n'est pas pertinent pour nos besoins car il ne contient aucune information que nous pouvons utiliser pour la classification [23].

Les meilleurs attributs ont les scores suivants (plus grand est le score plus la dépendance entre l'attribut et la valeur de sortie (qui est dans notre cas le type de flux) est plus forte) :

Attributs	Score
dst2src_packet_rate	$3.4149 \cdot 10^6$
src2dst_packet_rate	$2.5904 \cdot 10^7$
dst2src_inter_time_std	$5.7223 \cdot 10^8$
dst2src_packets	$9.8335 \cdot 10^6$
src2dst_inter_time_std	$8.1730 \cdot 10^8$
src2dst_inter_time_max	$2.9220 \cdot 10^9$
flow_use_time	$1.0988 \cdot 10^9$
flow_idle_time	$7.4455 \cdot 10^9$
Protocol	$5.0624 \cdot 10^5$
flow_duration	$8.3937 \cdot 10^9$
dst2src_inter_time_max	$2.4608 \cdot 10^9$
Packets	$6.9628 \cdot 10^6$
Bytes	$8.0217 \cdot 10^9$
dst2src_inter_time_avg	$2.4689 \cdot 10^8$
src2dst_inter_time_avg	$7.1017 \cdot 10^8$

Tableau 4.1 : Les attributs choisis et leurs scores attribués par la formule chi-square

Attribut SDN	Attribut HogZilla	Description
Duration	Flow duration	Durée depuis l'installation de flux dans la table des flux
Protocol	Protocol	Type de protocole (TCP, UDP, ICMP)
Bytes count	Bytes	Total d'octets transférés dans les deux sens
Packet count	Packets	Nombre total de paquets transférés dans les deux sens
Tx_packets	src2dst packets	Nombre d'octets de la source à la destination
Rx_packets	dst2src packets	Nombre d'octets de la destination à la source

Tableau 4.2 : Les attributs extraits du contrôleur et leurs équivalents dans HogZilla

Les six caractéristiques de jeu de données HogZilla présentées dans le tableau 4.2 peuvent être obtenues directement à partir du contrôleur SDN via des appels appropriés. En particulier, toutes ses caractéristiques sont lues à partir des statistiques de flux obtenus par l'appel OpenFlow (**OFP\_Flow\_stats\_request**) comme indiqué dans le chapitre 3 sur le module de collection du flux.

Dans le tableau 4.2 précédent, nous montrons le nom de l'attribut SDN et son équivalent dans HogZilla. À titre d'exemple, l'attribut concernant le protocole (TCP, UDP, ICMP) est utile pour la détection de Botnet car les bots utilisent généralement différents protocoles à différentes phases de leurs cycles de vie (configuration, attaque, mise à jour, etc.)

D'un autre côté, les 10 autres attributs HogZilla exploités et décrits dans le tableau 4.3 suivant peuvent être convenablement dérivés dans un environnement SDN comme suit:

No d'attribut	Attribut HogZilla	Description
7	src2dst_packets_rate	Nombre de paquets par seconde ( <b>PP : paquet per second</b> ) de la source à la destination
8	dst2src_packets_rate	Nombre de paquets par seconde ( <b>PP</b> ) de la destination à la source
9,10,11	src2dst_inter_time_{avg,max,std}	Différence de temps d'arrivée ( <b>IAT : Inter-Arrival Time</b> ) des paquets entre la source et la destination (moy, max, std)
12,13,14	dst2src_inter_time_{avg,max,std}	Différence de temps d'arrivée des paquets entre la destination et la source (moy, max, std)
15	Flow_use_time	Temps d'utilisation de flux
16	Flow_idle_time	Temps d'inactivité de flux

Tableau 4.3 : les attributs HogZilla calculés et dérivés dans un environnement SDN

$$PP = \frac{\text{nombre de paquets}}{\text{duration de flux}}$$

*flow use time = laps de temps entre le premier et le dernier paquet*

$$IAT = \frac{\text{flow use time}}{\text{nombre de paquets arrivés dans ce temps}}$$

*flow idle time = flow duration – flow use time*

Notez que nous exploitons en fait les attributs PP et IAT par rapport aux paquets échangés dans une direction spécifique (source vers destination et destination vers source). Avoir des attributs spécifiques à la direction est utile pour détecter des Botnets particuliers puisque, par exemple, la plupart des Botnets génèrent des flux unidirectionnels.

De plus, nous considérons le maximum, la moyenne et l'écart type de ces valeurs. En effet, le trafic de bot a tendance à être uniformément réparti dans le temps mais, au fur et à mesure qu'un bot progresse dans son cycle de vie et accomplit différentes tâches, les caractéristiques du trafic généré varient. L'utilisation de telles mesures statistiques permet de capturer les différences de comportement entre ces tâches, de sorte que le comportement global du bot puisse être détecté.

#### 4.4 Adaptation du réseau de neurones au jeu de données HogZilla :

La phase d'apprentissage a pour but de produire un modèle avec un haut taux de classifications correctes, pour cela nous avons adapté notre réseau de neurones approfondi MLP au jeu de données HogZilla.

Le réseau de neurones profond utilisé est caractérisé par les points suivants :

- La couche d'entrée est constituée d'un certain nombre fixe de neurones selon le nombre de caractéristiques. Ces caractéristiques sont sélectionnées à partir d'un jeu de données utilisée dans la phase d'apprentissage (dans notre cas nous avons utilisé 16 neurones pour le dataset Hogzilla).
- Il y a 3 couches cachées avec une distribution unique de neurones : nous avons utilisé 6 neurones par couche qui sont activés en utilisant la fonction d'activation ReLU (Unité de rectification linéaire).
- La couche de sortie se compose d'un seul neurone avec une fonction d'activation sigmoïde.

Nous avons utilisé la fonction d'activation ReLU qui fonctionne généralement mieux que d'autres fonctions d'activation pour les couches cachées. L'augmentation des performances est due au fait que la fonction d'activation ReLU est une fonction linéaire non saturante.

Contrairement aux fonctions d'activation sigmoïde / logistique ou hyperbolique tangente, le ReLU ne sature pas à -1, 0 ou 1. Une fonction d'activation saturante se converge vers une valeur. La fonction tangente hyperbolique, par exemple, converge vers -1 lorsque  $x$  diminue et à 1 lorsque  $x$  augmente car l'ajustement des poids dans la phase de rétro-propagation s'appuie sur le calcul de dérivé de la fonction d'activation, ce qui pose un problème pour la fonction sigmoïde et hyperbolique tangente comme

les valeurs du dérivé vont être considérablement petites (toujours  $< 0.25$ ). Ceci provoque un changement insignifiant des poids et par conséquent il risque de ne pas y avoir une amélioration dans les performances du réseau de neurones.

En effet, la fonction d'activation sigmoïde est couramment utilisée dans la dernière couche des réseaux de neurones dans les classifications binaires afin de transformer les résultats en probabilités qui indiquent les classes. Dans notre cas, les valeurs de sortie possibles sont 1 pour le trafic Botnet et 0 pour le trafic normal.

À partir d'une telle architecture de base, nous avons essayé plusieurs paramètres de configuration différents pour le réseau de neurones, par exemple nombre de périodes d'apprentissage (époques) et tailles du lot (**batch size**), et nous avons fixé ces valeurs à 100 pour le nombre d'époques et 128 pour la taille du lot. Les résultats des différentes configurations testés seront abordées dans le chapitre 5.

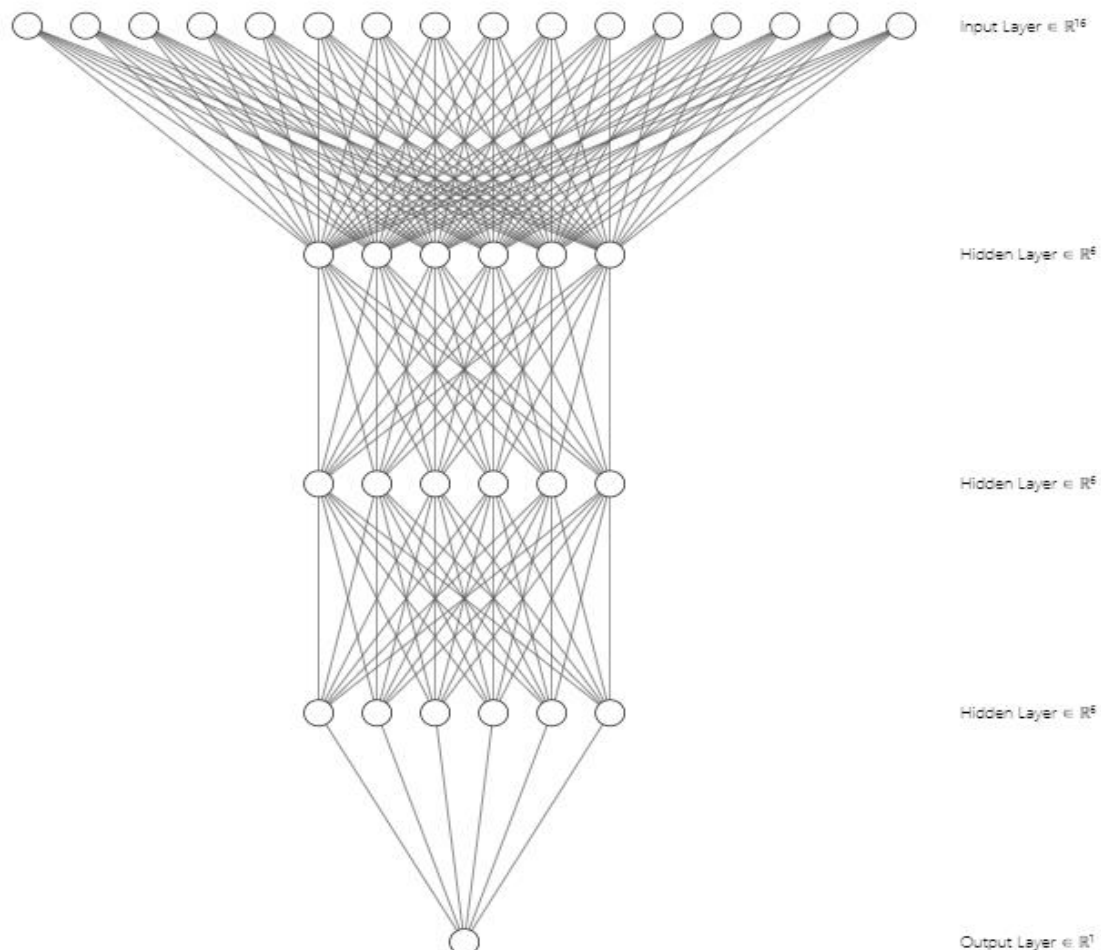


Figure 4.1 : Le réseau de neurones utilisé dans notre approche

## 4.5 Langage de programmation et librairies utilisés :

### 4.5.1 Python :

Python est l'un des langages de programmation les plus populaires pour la science des données (**Data science**) et bénéficie donc d'un grand nombre de bibliothèques complémentaires utiles couvrant les types d'apprentissage automatique et profond, développées par sa communauté open-source que nous citer ensuite par la suite [13].

### 4.5.2 Tensorflow :

Il s'agit d'une bibliothèque qui peut être utilisée pour créer des modèles d'apprentissage profond. Tensorflow peut fonctionner sur des systèmes à processeur unique, des GPU ainsi que des appareils mobiles et des systèmes distribués à grande échelle de centaines de machines [29].

### 4.5.3 Keras :

Keras est une bibliothèque Python pour l'apprentissage profond qui peut s'exécuter au-dessus de TensorFlow. Elle a été développée pour rendre le développement de modèles d'apprentissage profond aussi rapide et facile comme TensorFlow peut être difficile à utiliser directement pour créer ces modèles. Il fonctionne sur Python 2.7 ou 3.5 et peut s'exécuter parfaitement sur les GPU ainsi que les CPU [29].

### 4.5.4 NumPy :

NumPy permet d'effectuer des opérations mathématiques sur les structures de données souvent utilisées en Machine Learning : vecteurs, matrices et les manipuler [23].

### 4.5.5 Scikit-learn :

La bibliothèque Scikit-learn se concentre sur le développement des modèles basés sur les algorithmes d'apprentissage automatique pour la classification, la régression, le clustering et plus encore. Il fournit également des outils pour les tâches connexes telles que l'évaluation des modèles, le réglage des paramètres et le prétraitement des données [30].

#### 4.5.6 Pandas :

La bibliothèque contient des outils et structures de données pour organiser et analyser les données [30].

### 4.6 Outils et logiciels de développements :

#### 4.6.1 Jupyter Notebook :

Jupyter Notebook est une application Web open source qui vous permet de créer et de partager des documents contenant du code écrit en direct, des équations, des visualisations et du texte narratif. Les utilisations comprennent : le nettoyage et la transformation des données, la simulation numérique, la modélisation statistique, la visualisation des données, l'apprentissage automatique et bien plus encore.

Jupyter prend en charge plus de 40 langages de programmation, dont **Python**, **R**, **Julia** et **Scala** [31].

### 4.7 Résultats numériques :

#### 4.7.1 Matrice de confusion (Confusion matrix) :

La matrice de confusion est une présentation pratique de l'exactitude d'un modèle avec deux classes ou plus. La matrice présente les valeurs réelles sur l'axe des x et les résultats de prédiction sur l'axe des y. Les cellules de la matrice représentent le nombre de prédictions faites par un algorithme d'apprentissage automatique. Par exemple, un algorithme d'apprentissage automatique peut prédire 0 ou 1 et chaque prédiction peut en fait avoir une valeur réelle de 0 ou 1. Les prédictions pour 0 qui étaient en fait 0 apparaissent dans la cellule pour prédiction = 0 et réel = 0, tandis que les prédictions pour 0 qui étaient en fait 1 apparaît dans la cellule pour prédiction = 0 et réel = 1 (**Les valeurs qui sont correctement prédites se retrouvent à la diagonale de la matrice**). [30]



		Les valeurs prédites	
		Classe positive	Classe négative
Les valeurs réelles	Classe positive	VP	FN
	Classe négative	FP	VN

Tableau 4.4 : La matrice de confusion

Où :

- *VP* est le nombre de vrais positifs. Des observations qui font partie de la classe positive (une personne a une maladie, a acheté le produit, etc.) et que le modèle a prédit correctement.
- *VN* est le nombre de vrais négatifs. Des observations qui font partie de la classe négative (une personne n'a pas la maladie, n'a pas acheté le produit, etc.) et que le modèle a prédit correctement.
- *FP* est le nombre de faux positifs. Également appelée erreur de type I. Le nombre d'observations prédites d'être de la classe positive qui font en fait partie de la classe négative.
- *FN* est le nombre de faux négatifs. Également appelée erreur de type II. Le nombre d'observations prédites d'être de la classe négative qui font en fait partie de la classe positive.

#### 4.7.2 Métriques d'évaluation :

**Exactitude (Accuracy) :** L'exactitude est le nombre de prédictions correctes faites par rapport à toutes les prédictions faites [23].

$$Exactitude = \frac{VP + VN}{VP + FP + VN + FN}$$

**Précision (Precision) :** La précision est la proportion des observations prédites être positive et qui ont en fait positive.

Les modèles à haute précision sont pessimistes dans le sens où ils ne prédisent qu'une observation est de classe positive uniquement lorsqu'ils en sont très sûrs [23].

$$Precision = \frac{VP}{VP + FP}$$

**Rappel (Recall) :** Le rappel est le nombre d'instances de la classe positive qui ont été prédit correctement.

Les modèles avec un rappel élevé sont optimistes dans le sens où ils ont une barre basse pour prédire qu'une observation est dans la classe positive [23].

$$Rappel = \frac{VP}{VP + FN}$$

**F-Mesure (F1-score) :** Le score F1 est la moyenne harmonique de la précision et du rappel, où un score F1 atteint sa meilleure valeur à 1 (précision et rappel parfaits) [23].

$$F1 - score = 2 * \frac{Rappel * Precision}{Rappel + Precision}$$

#### 4.7.3 Evaluation des performances :

Pour évaluer les performances de notre réseau de neurones, nous avons fixé le jeu de données de test à 20% et le jeu de données d'apprentissage à 80%. De plus, comme nous l'avons déjà spécifié dans ce chapitre, nous avons fixé la taille du lot à 128 et le nombre d'époque à 100 car nous avons obtenu des meilleures performances avec ces paramètres. La comparaison entre les différents résultats des tests pour choisir les meilleurs paramètres sont indiqués dans le prochain chapitre.

Les métriques sont calculées à l'aide de la validation croisée à k-pli (**K-fold cross validation**).

La validation croisée est une approche qui peut être utilisée pour estimer les performances d'un algorithme d'apprentissage automatique ou profond. Cela fonctionne en divisant le jeu de données en k parties (par exemple, k = 5 ou k = 10). Chaque partie des données est appelée un pli (**fold**). L'algorithme est entraîné (formé) sur k-1 plis avec un pli retenu et le modèle est testé sur le pli retenu. Ceci est répété afin que chaque pli de jeu de données ait une chance d'être un jeu de test retenu. Après avoir exécuté la validation croisée, k scores de performance différents sont obtenus, ces derniers vont être résumés à l'aide d'une moyenne qui représente un résultat précis. La validation croisée est plus précise car l'algorithme est entraîné et évalué plusieurs fois sur différentes données.

Les résultats obtenus par la validation croisée à **5-plis** sont indiqués dans la matrice de confusion suivante et le tableau 4.6

		Les valeurs prédites	
		Botnet	Normal
Les valeurs réelles	Botnet	<b>63885 VP</b>	2125 FN
	Normal	1215 FP	<b>64997 VN</b>

Tableau 4.5 : Matrice de confusion avec Botnet est la classe positive et trafic normal est la classe négative

Métriques	Résultat
<b>Exactitude</b>	97,47%
<b>Précision</b>	96,83%
<b>Rappel</b>	98,16%
<b>F-mesure</b>	97,49%

Tableau 4.6 : Les résultats obtenus dès la validation de notre approche au HogZilla dataset

Après la validation de notre approche sur le jeu de données HogZilla, nous avons obtenu des résultats satisfaisants avec plus de 97% de classifications correctes, plus de 96% pour la précision et 98% comme rappel, nous passerons à l'étape simulation des attaques Botnet dans un réseau SDN.

## 4.8 Conclusion :

Dans ce chapitre, nous avons présenté le jeu de données utilisé ainsi que les attributs choisis pour une détection adéquate de trafic Botnet, nous avons énuméré les libraires utilisées et les outils de développements et enfin nous avons montré les résultats obtenus.

Dans le dernier chapitre, nous allons présenter la partie simulation ainsi que l'environnement de travail déployé avant de comparer les différents résultats des paramètres testés pour avoir des mesures optimales. Pour finir, une comparaison sera faite entre les résultats des différents algorithmes de Machine Learning et notre approche Deep Learning.

# Chapitre 5 : Simulation Mininet et étude comparative des différents résultats

## 5.1 Introduction :

Dans ce dernier chapitre, nous présenterons l'environnement de travail matériel et logiciel pour la partie simulation, nous énumérons les différents logiciels utilisés pour cette partie et nous clôturerons par une étude comparative des différents résultats de multiple configurations essayées et les différents algorithmes d'apprentissage automatique testés ainsi que les résultats de notre travail avec ceux des travaux antérieurs cités dans le deuxième chapitre.

## 5.2 Environnement de travail :

### 5.2.1 Environnement matériel :

Voici ci-dessous les caractéristiques de la machine utilisée dans la simulation des attaques Botnet

**Processeur :** 4th Gen Intel® Core™ i5-4300U @ 2.90 GHz

**RAM :** 4,00 GO

**GPU :** Intel HD Graphics 4400 1,50 GO

### 5.2.2 Environnement logiciel :

Nous citerons les différents outils, bibliothèques et logiciels utilisés dans la phase de simulation :

**Ubuntu :** Ubuntu est une distribution du système d'exploitation Linux basée sur Debian principalement composée de logiciels libres et open source. Ubuntu est officiellement publié en trois éditions : Desktop, Server et Core. Toutes les éditions peuvent s'exécuter sur un ordinateur ou dans une machine virtuelle. Ubuntu est un système d'exploitation populaire aussi pour le Cloud Computing [32].

**Ryu controller :** Ryu est une infrastructure logicielle (**Framework**) de réseau défini par logiciel (SDN) basé sur les composants. Ryu fournit des composants logiciels avec une API bien définie qui permettent aux développeurs de créer facilement de nouvelles

applications de gestion et de contrôle de réseau SDN. Ryu prend en charge divers protocoles de gestion des équipements réseau, tels que OpenFlow, Netconf, OF-config, etc. À propos d'OpenFlow, Ryu prend entièrement en charge les versions 1.0, 1.2, 1.3, 1.4, 1.5 et Nicira. Ryu est disponible gratuitement sous la licence Apache 2.0 [33].

### **Mininet :**

Mininet est un outil de prototypage rapide de réseaux définis par logiciel (SDN) il permet de [34] :

- Créer un réseau virtuel réaliste avec de vrais composants.
- Offrir la possibilité de créer des hôtes, commutateurs et des contrôleurs via :
  - Ligne de commande
  - Interface utilisateur
  - Application Python

### **Scapy :**

Scapy est un logiciel libre de manipulation de paquets réseaux écrit en python. Il est capable, entre autres, d'intercepter le trafic sur un segment réseau, de générer des paquets dans un nombre important de protocoles et d'analyser le réseau.

Ces capacités permettent la construction d'outils qui peuvent sonder, scanner ou attaquer les réseaux [35].

### **IPERF :**

Iperf est un outil largement utilisé pour la mesure et le réglage des performances du réseau. Iperf a des fonctionnalités client et serveur et peut créer des flux de données pour mesurer le débit entre les deux extrémités dans un ou les deux sens. Les flux de données peuvent être soit du protocole (TCP) ou (UDP) [36].

### **Xterm terminal :**

Xterm est un émulateur de terminal. Un utilisateur peut disposer de plusieurs instances de Xterm simultanément dans le même écran, chacune d'entre elles offrant des entrées/sorties indépendantes pour les processus qui s'y exécutent [38].

## 5.3 La Simulation des attaques Botnet dans un environnement

SDN :

### 5.3.1 Topologie Mininet :

Mininet offre la possibilité de créer des réseaux SDN virtuels en écrivant un script python ou grâce aux lignes de commandes, et comme la création de topologie par un code écrit en python est meilleure parce qu'elle nous donne le contrôle total sur l'emplacement des différents équipements réseaux dans la topologie, nous avons choisi cette dernière.

Notre réseau consiste en deux commutateurs (switch), un contrôleur et six machines où les cinq premières machines sont reliées au premier switch et la dernière machine est reliée au deuxième switch, et les deux commutateurs sont reliés au contrôleur.

Nous nous sommes concentrés dans notre simulation sur les attaques DDOS (Distributed Denial Of Service) car les Botnets sont généralement utilisés pour ce but. La topologie consiste alors en un réseau Botnet client/serveur, où la première machine sera le maître (**Botmaster**), l'ensemble de deuxième, troisième, quatrième et cinquième seront les esclaves (**Botslaves**) et la dernière machine sera la victime des attaques DDOS.

La figure 5.1 ci-dessous donne un aperçu sur la topologie construite :

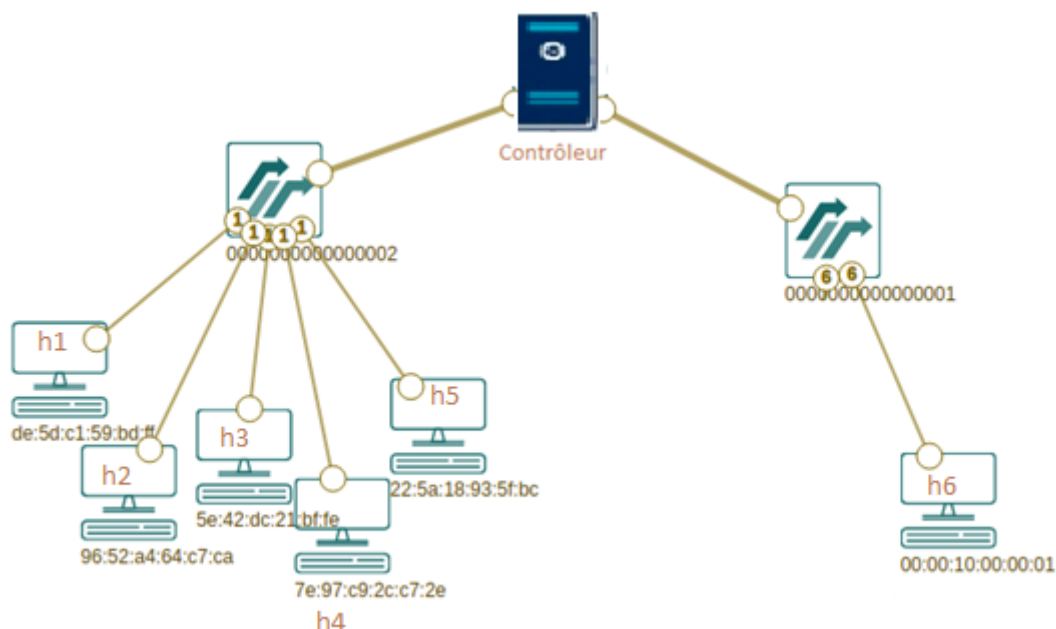


Figure 5.1 : La topologie construite avec l'outil Mininet

Où :

- L'hôte **h1** est le maître, celui-ci exécute un script python *master.py* pour établir une connexion client/serveur avec les hôtes esclaves, il transmet ensuite l'adresse et le numéro de port de la victime aux esclaves afin d'effectuer l'attaque.
- Les hôtes **h2, h3, h4, h5** sont les hôtes esclaves qui exécutent le script python *slave.py* pour établir une connexion client/serveur avec l'hôte maître et attaquer l'hôte victime selon l'ordre reçu.
- L'hôte **h6** est l'hôte victime des attaques DDOS, cet hôte est soit un serveur TCP ou UDP ou un hôte ordinaire qui reçoit les requêtes PING selon le type d'attaque.

### 5.3.2 Les attaques DDOS :

Dans notre simulation, trois types d'attaques DDOS ont été pris en charge, un type par protocole (TCP, UDP et ICMP), les trois sont des attaques par inondation (**Flooding**) :

**TCP SYN Flood** : Un attaquant SYN flood envoie uniquement les messages SYN sans répondre (pas d'envoi d'un message d'acquittement ACK) à la réponse (SYN et ACK) du serveur TCP. Le protocole TCP oblige le serveur à allouer un bloc de mémoire appelé bloc de contrôle et à attendre un certain temps avant d'abandonner la connexion TCP. Si l'attaquant envoie des milliers de messages SYN, le serveur doit mettre les messages en file d'attente et attendre le temps requis avant de les effacer et de libérer toute mémoire associée. Une fois que la mémoire tampon pour stocker ces messages SYN est pleine, le serveur peut ne plus être en mesure de recevoir plus de messages SYN. [37]

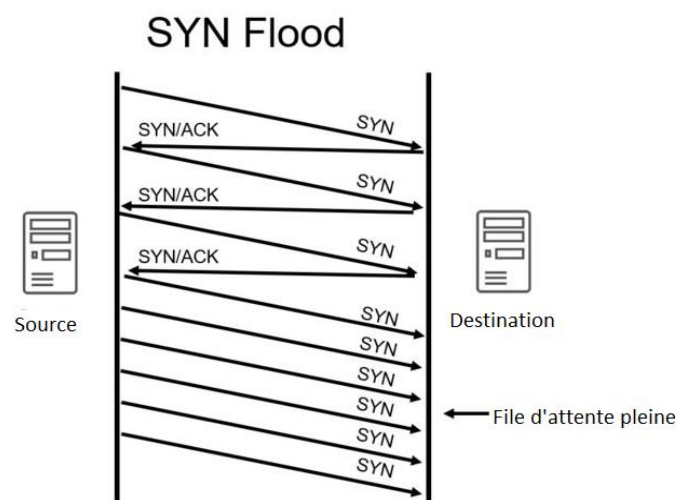


Figure 5.2 : Exemple d'attaque SYN FLOOD



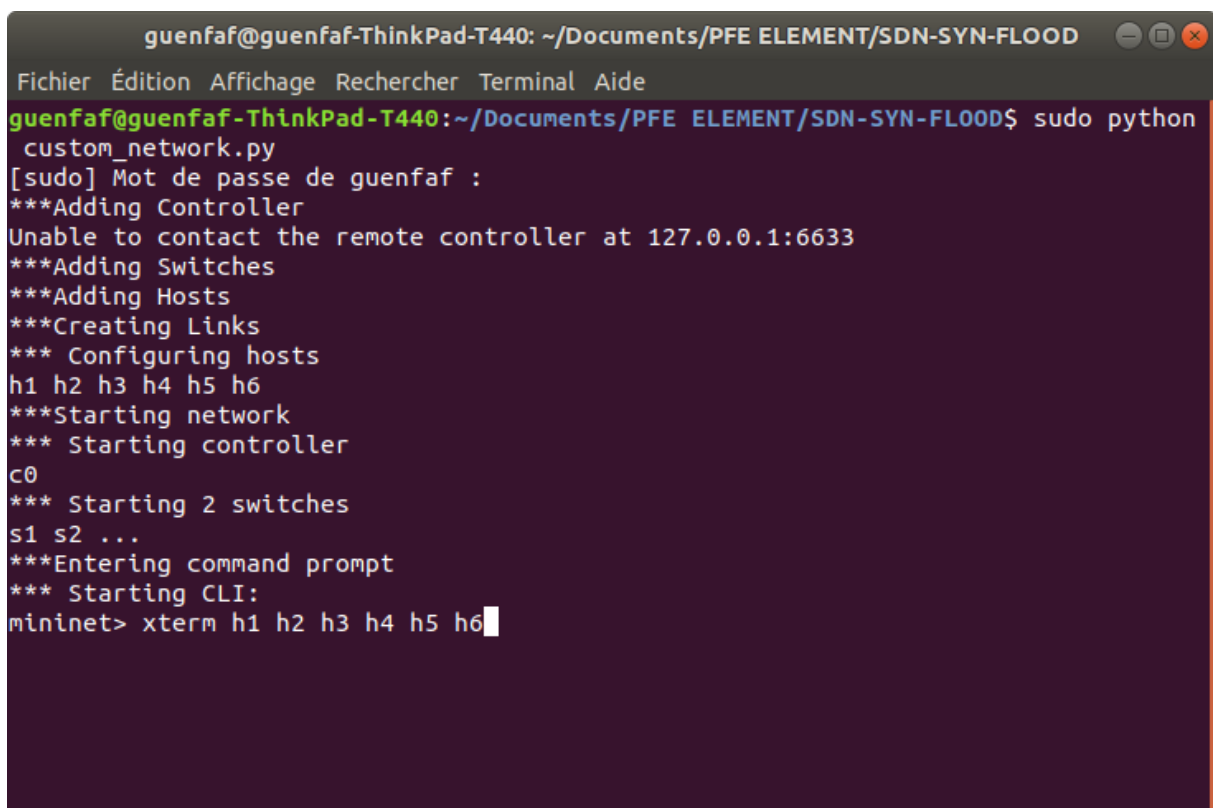
**UDP Flood :** Dans une attaque UDP Flood, l'attaquant envoie un grand nombre de petits paquets UDP vers des ports aléatoires (ou spécifiques) de l'hôte victime. Chaque paquet nécessite du temps de traitement, de la mémoire et de la bande passante. Si l'attaquant envoie suffisamment de paquets, L'attaque neutralise la capacité de traitement et de réponse du serveur UDP [37].

**ICMP Flood :** La plupart des attaques ICMP Flood sont basées sur ICMP Echo et appelées attaques d'inondation PING. Dans Ping flood le pirate tente de submerger un appareil ciblé avec des paquets de demande d'écho ICMP, rendant la cible inaccessible au trafic normal [8].

Après avoir cité les différents types d'attaques pris en charge, nous expliquerons les différentes étapes du déroulement de la simulation des attaques :

### Etape 1 : création de la topologie avec un script python

Un script python est préparé et ensuite exécuté pour créer notre réseau SDN avec la commande : `sudo python custom_network.py`. La figure 5.3 ci-dessous montre l'étape de la création :



```
guenfaf@guenfaf-ThinkPad-T440: ~/Documents/PFE ELEMENT/SDN-SYN-FLOOD
Fichier Édition Affichage Rechercher Terminal Aide
guenfaf@guenfaf-ThinkPad-T440:~/Documents/PFE ELEMENT/SDN-SYN-FLOOD$ sudo python
custom_network.py
[sudo] Mot de passe de guenfaf :
***Adding Controller
Unable to contact the remote controller at 127.0.0.1:6633
***Adding Switches
***Adding Hosts
***Creating Links
*** Configuring hosts
h1 h2 h3 h4 h5 h6
***Starting network
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
***Entering command prompt
*** Starting CLI:
mininet> xterm h1 h2 h3 h4 h5 h6
```

Figure 5.3 : La création de topologie SDN par outil Mininet

## Etape 2 : Démarrage des instances Xterm

Pour chaque hôte, un terminal doit être démarré à l'aide de la commande `xterm h1 h2 h3 h4 h5 h6`. Cette commande démarre six terminaux dont chacun représente un hôte.

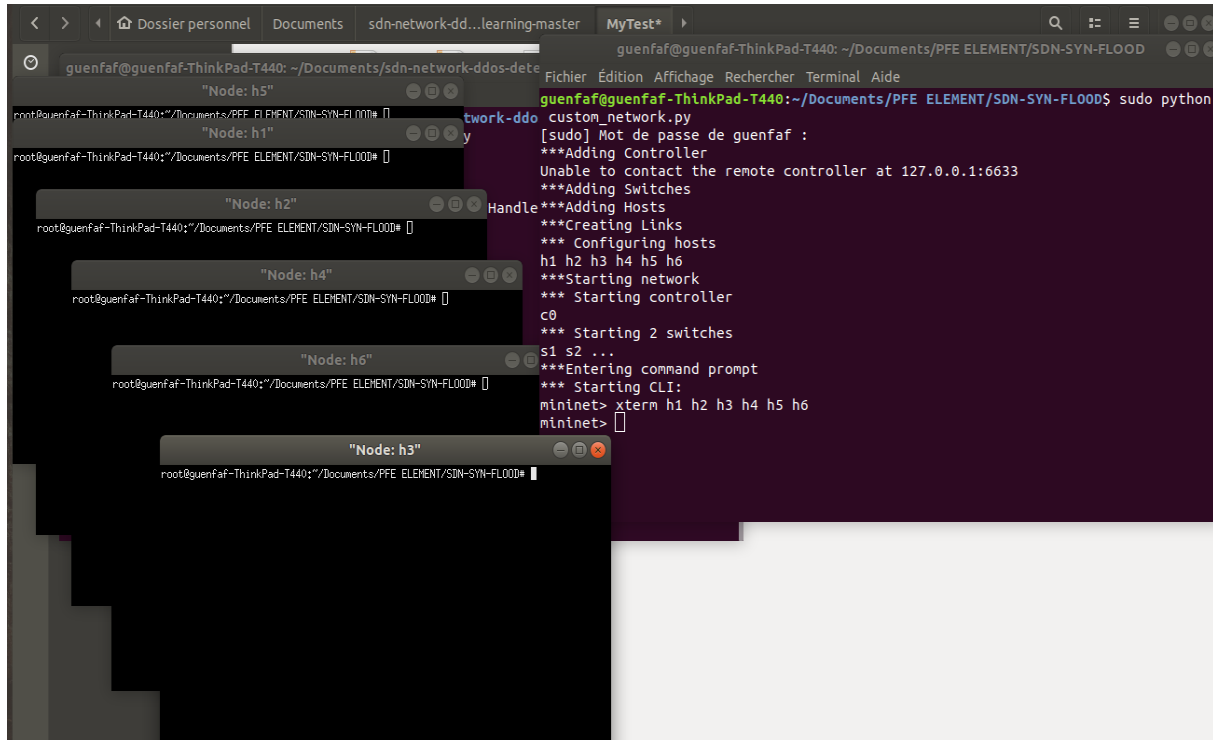


Figure 5.4 : Les six instances des six hôtes d'émulateur de terminal Xterm

## Etape 3 : L'exécution de code maître (Botmaster) et esclave (Botslaves)

Comme l'hôte h1 est le maître, celui-ci exécutera son code approprié par commande `python master.py` dans le but d'établir une connexion client /serveur avec ses esclaves par l'utilisation des sockets TCP, alors que les hôtes esclaves essayeront de se connecter au Botmaster en exécutant le code `slave.py` par la commande `python slave.py`.

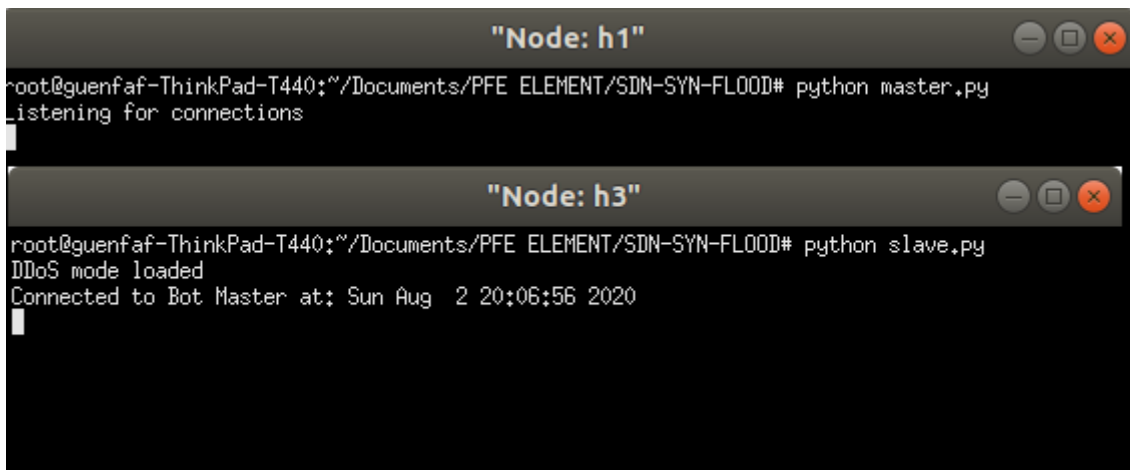


Figure 5.5 : L'établissement de connexion entre le Bot maître et les esclaves

### Etape 3 : Démarrage de serveur TCP/UDP

L'hôte h6 sera l'hôte victime des attaques DDOS qui sera un serveur TCP ou UDP selon le scénario du déroulement de la simulation :

**Cas 1 : TCP FLOOD :** L'outil IPERF sera utilisé pour démarrer un serveur TCP par la ligne de commande *iperf -s -p 5201*. Le serveur va écouter pour recevoir des requêtes TCP SYN sur le port 5201

**Cas 2 : UDP FLOOD :** L'outil IPERF sera utilisé pour démarrer un serveur UDP par la ligne de commande *iperf -u -s -p 5201*. Le serveur va écouter pour recevoir des datagrammes UDP sur le port 5201

**Cas 3 : ICMP FLOOD :** l'hôte victime reçoit des paquets ICMP Echo, le démarrage d'un serveur TCP/UDP n'est pas nécessaire.

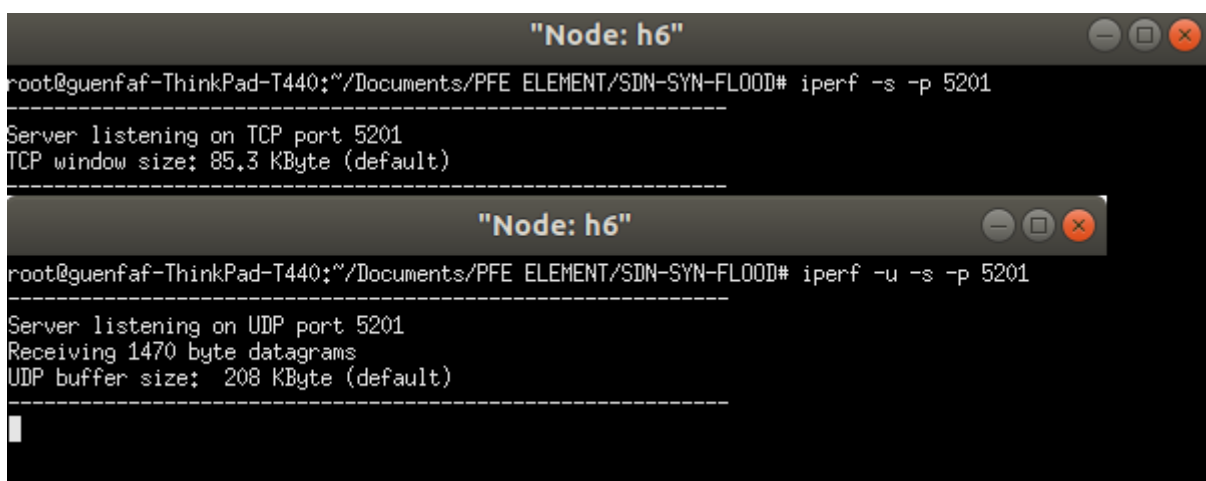
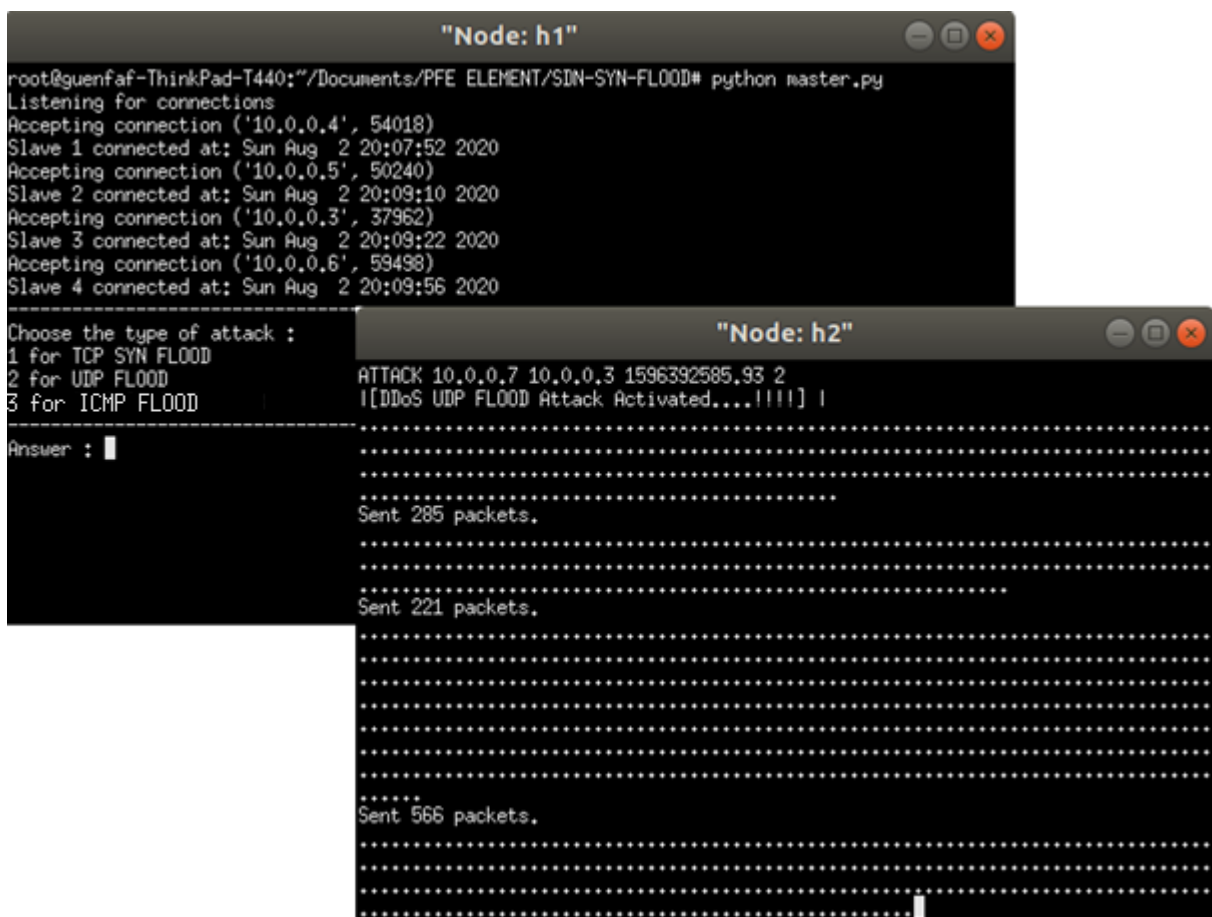


Figure 5.6 : Démarrage du serveur victime par l'outil IPERF

#### Etape 4 : Inondation du serveur victime par le flux DDOS

Une fois la connexion entre l'hôte maître et les hôtes esclaves est établie, le maître envoie l'adresse, le numéro de port de la victime ainsi que le type d'attaque d'inondation choisi parmi 3 variantes (TCP FLOOD, UDP FLOOD ou ICMP FLOOD) aux esclaves.

Les différents esclaves initient l'attaque lorsque l'ordre du maître est reçu.



```
"Node: h1"
root@guenaf-ThinkPad-T440:~/Documents/PFE ELEMENT/SDN-SYN-FLOOD# python master.py
Listening for connections
Accepting connection ('10.0.0.4', 54018)
Slave 1 connected at: Sun Aug 2 20:07:52 2020
Accepting connection ('10.0.0.5', 50240)
Slave 2 connected at: Sun Aug 2 20:09:10 2020
Accepting connection ('10.0.0.3', 37962)
Slave 3 connected at: Sun Aug 2 20:09:22 2020
Accepting connection ('10.0.0.6', 59498)
Slave 4 connected at: Sun Aug 2 20:09:56 2020

Choose the type of attack :
1 for TCP SYN FLOOD
2 for UDP FLOOD
3 for ICMP FLOOD
Answer : 2

"Node: h2"
ATTACK 10.0.0.7 10.0.0.3 1596392585.93 2
[DDoS UDP FLOOD Attack Activated....!!!!]
.....
Sent 285 packets.
.....
Sent 221 packets.
.....
Sent 566 packets.
.....
```

Figure 5.7 : Inondation du serveur UDP par les esclaves

#### Etape 5 : Démarrage du contrôleur Ryu avec le système de détection des attaques

Le contrôleur Ryu est démarré en tapant la commande : *ryu-manager controller-ANN.py*. Ce script contient les fonctionnalités principales pour un contrôleur dans l'environnement SDN tel que l'établissement des tables de flux, les mises à jour.....etc. ainsi que le système établi pour détecter les attaques DDOS Botnet.

```
guenfaf@guenfaf-ThinkPad-T440: ~/Documents/sdn-network-ddos-detection-using-machin...
Fichier Édition Affichage Rechercher Terminal Aide
guenfaf@guenfaf-ThinkPad-T440:~/Documents/sdn-network-ddos-detection-using-machin...$ ryu-manager controller-ANN.py
loading app controller-ANN.py
loading app ryu.controller.ofp_handler
instantiating app controller-ANN.py of SimpleMonitor13
2020-08-02 20:12:28.652997: I tensorflow/core/platform/cpu_feature_guard.cc:142]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
2020-08-02 20:12:28.698588: I tensorflow/core/platform/profile_utils/cpu_utils.c
c:94] CPU Frequency: 2494200000 Hz
2020-08-02 20:12:28.699104: I tensorflow/compiler/xla/service/service.cc:162] XL
A service 0x5563e07df470 executing computations on platform Host. Devices:
2020-08-02 20:12:28.699162: I tensorflow/compiler/xla/service/service.cc:169]
StreamExecutor device (0): <undefined>, <undefined>
----- Model loaded successfully... -----
instantiating app ryu.controller.ofp_handler of OFPHandler
```

Figure 5.8 : Initialisation de contrôleur RYU avec le script de détection

## Étape 5 : Détection des attaques DDOS

Le contrôleur Ryu envoie périodiquement des requêtes pour recevoir la table de flux des différents commutateurs dans le but d'analyser le trafic réseau comme il est expliqué au chapitre 3.

Lorsque le système détecte un trafic malin, un message est affiché pour indiquer la présence des flux malicieux.

```
guenfaf@guenfaf-ThinkPad-T440: ~/Documents/sdn-network-ddos-detection-using-machin...
Fichier Édition Affichage Rechercher Terminal Aide
* DDOS traffic :52508
-----
Warning BOTNET activity detected ...
Accuracy_score :96.68%
* total traffic :74102
* normal traffic :2460
* DDOS traffic :71642
-----
Warning BOTNET activity detected ...
Accuracy_score :97.16%
* total traffic :76442
* normal traffic :2168
* DDOS traffic :74274
-----
Warning BOTNET activity detected ...
Accuracy_score :97.49%
* total traffic :79897
* normal traffic :2004
* DDOS traffic :77893
-----
```

Figure 5.9 : Un aperçu sur l'application de détection des attaques Botnet

## 5.4 Etude comparative des différents résultats :

### 5.4.1 Comparaison des différentes configurations :

Pour choisir les meilleurs paramètres des différentes configurations testées, il faut les comparer selon les métriques Exactitude, Précision, Rappel et F1-Mesure (paragr. 4.6.1). Pour cela les tableaux suivant montrent les résultats des comparaisons effectuées.

Les différentes configurations concernent les paramètres suivants :

- Nombre de couches cachées dans le réseau de neurones
- La taille du lot (Batch size)
- Nombre d'époques

Nombre d'époques	Exactitude	Précision	Rappel	F1-mesure
10	95.93%	94.64%	97.40%	96.00%
20	96.46%	96.19%	96.76%	96.47%
50	96.85%	97.41%	96.27%	96.84%
<b>100</b>	<b>97.35%</b>	<b>96.76%</b>	<b>97.99%</b>	<b>97.37%</b>
200	97.42%	96.70%	98.11%	97.40%

Tableau 5.1 : Les résultats obtenus par les différents nombres d'époques testés

Un petit nombre d'époques signifie une période d'apprentissage insuffisante pour des performances supérieures, ce qui explique l'infériorité des résultats en utilisant 10 époques par rapport à celles de 20 époques, de 20 époques à celles de 50 et de 50 par rapport à 100. Après 100 époques nous avons remarqué que les résultats convergent vers une valeur, l'erreur atteint sa valeur minimale et il n'y a pas une amélioration significative des résultats, pour cela nous avons pris la valeur 100 comme un nombre idéal d'époques.

Taille du lot	Exactitude	Précision	Rappel	F1-mesure
16	95.23%	94.90%	95.61%	95.25%
32	95.85%	94.38%	97.52%	95.93%
<b>128</b>	<b>96.26%</b>	<b>96.56%</b>	<b>95.94%</b>	<b>96.25%</b>
256	94.85%	92.28%	97.90%	95.01%
1024	92.48%	95.57%	89.11%	92.23%
4096	91.31%	88.02%	95.68%	91.69%

Tableau 5.2 : Les résultats obtenus par les différentes tailles du lot testées

Le tableau 5.2 montre que la taille du lot a un impact significatif sur l'apprentissage. En général, des lots de plus grande taille engendrent un apprentissage plus rapide, mais n'obtiennent pas toujours des résultats acceptables.

Les lots plus petits s'entraînent plus lentement, mais obtiennent des précisions plus élevées comme le nombre d'itérations est plus élevé.

Une taille du lot de 16, 32 ou 128 résulte en performance meilleure qu'une taille de 256, 1024 ou 4096 car l'erreur résultant dans des lots de petite taille est inférieure à celle des lots de grande taille. Un facteur important est la fonction de calcul d'erreur et la mise à jour des poids. Dans un lot de 128 le calcul d'erreur et l'ajustement des poids se fait plus fréquemment que dans un lot de 256 ou 1024 grâce à l'augmentation du nombre d'itérations qui signifie un bon ajustement des poids.

$$\text{Nombre d'itérations} = \frac{\text{Taille du jeu de données}}{\text{Taille du lot}} * \text{Nombre d'époques}$$

Nombre de couches cachées	Taille du lot	Nombre d'époque	Exactitude	Précision	Rappel	F1-mesure
2	128	100	95.96%	94.94%	97.11%	96.01%
2	32	20	95.73%	93.83%	97.91%	95.83%
2	16	10	93.15%	96.59%	89.43%	92.90%
<b>3</b>	<b>128</b>	<b>100</b>	<b>97.47%</b>	<b>96.83%</b>	<b>98.16%</b>	<b>97.49%</b>
3	32	20	95.29%	96.49%	94.01%	95.23%
3	16	10	93.20%	94.99%	91.25%	93.08%
4	128	100	97.17%	97.36%	96.98%	97.17%
4	32	20	96.44%	95.85%	97.08%	96.46%
4	16	10	94.00%	90.85%	97.87%	94.23%

Tableau 5.3 : Les résultats obtenus par les différentes configurations testées

Un grand nombre de couches peut engendrer une dégradation des performances car un nombre important de couches peut provoquer un sur-apprentissage (**Overfitting**) : le modèle est en train de mémoriser des données d'apprentissage sans trouver la corrélation entre les entrées et la sortie. Ceci explique la détérioration des résultats en utilisant 4 couches cachées par rapport à 3.

D'après les résultats obtenus, il est évident qu'une configuration contenant 3 couches cachées, une taille du lot de 128 et 100 époques est idéale pour notre solution.



### 5.4.2 Comparaison des différents algorithmes de Machine Learning et Deep Learning :

Les résultats des tableaux ci-dessous montrent la fiabilité de l'apprentissage profond par rapport aux autres algorithmes d'apprentissage automatique avec un taux élevé de détection d'attaques Botnet DDOS.

Pour la classification des flux bénins tous les algorithmes ont montré de bons résultats sauf le SVM.

Algorithme	TCP Flood	UDP Flood	ICMP Flood
<b>DNN</b>	<b>97%-99%</b>	<b>96%-97%</b>	<b>86%-95%</b>
Random Forest	78%-81%	80%-86%	88%-97%
KNN	13%-31%	17%-23%	15%-22%
SVM	2%-5%	2%-5%	2%-5%
Naïve Bayes	95%-98%	97%-100%	20%-40%

Tableau 5.4 : Tableau comparatif des différents résultats obtenus par les algorithmes d'apprentissage automatique et profond dans un environnement d'attaques

Algorithme	TCP normal	UDP normal	ICMP normal
<b>DNN</b>	<b>95%-98%</b>	<b>97%-100%</b>	<b>97%-100%</b>
Random Forest	95%-98%	95%-97%	97%-100%
KNN	94%-98%	95%-97%	97%-100%
SVM	97%-100%	97%-100%	97%-100%
Naïve Bayes	58%-65%	97%-100%	54%-83%

Tableau 5.5 : Tableau comparatif des différents résultats obtenus par les algorithmes d'apprentissage automatique et profond dans un environnement ordinaire

### 5.4.3 Comparaison de notre approche et travaux antérieurs :

<b>Approche</b>	<b>Exactitude</b>	<b>Précision</b>	<b>Rappel</b>	<b>F1-mesure</b>	<b>Taux de faux négatif</b>	<b>Taux de faux positif</b>
Notre approche	97.47%	96.83%	98.16%	97.49%	1.87%	3.17%
Livadas et al	Non mentionné	Non mentionné	Non mentionné	Non mentionné	7.89%	15.04%
Beigi et al	75%	Non mentionné	Non mentionné	Non mentionné	Non mentionné	Non mentionné
Farhan et al	80%	Non mentionné	Non mentionné	Non mentionné	27.39%	12.81%
Tang et al	75.75%	83%	75%	74%	Non mentionné	Non mentionné
Aboubakar et al	97.32%	94.10%	Non mentionné	Non mentionné	0.71%	3.91%

Tableau 5.6 : Tableau comparatif des différents travaux

Le tableau 5.6 ci-dessus a montré l'efficacité de notre système de détection des attaques Botnets par rapport à certains travaux antérieurs cités dans la littérature (paragr. 2.10).

Le travail de Livadas et al. n'a pas mentionné les résultats obtenus sauf pour le taux de faux négatifs 7.89% et le taux de faux positifs 15.04% qui sont considérés comme assez élevés si nous prenons en compte le fait que les flux Botnets sont nombreux. Avec ces taux de faux négatifs et faux positifs, un nombre important des flux Botnets vont être mal classé ce qui menace la sécurité du réseau ou déclenche de fausses alarmes. Pour notre cas nous avons obtenus un taux meilleur de 1.87% pour le taux de faux négatif et 3.17% pour celui de faux positif.

Le travail de Baigi et al. a montré une exactitude acceptable de 75% mais un système de détection des attaques Botnet doit avoir un taux de détection plus élevé. Une explication possible est l'utilisation de l'algorithme de l'arbre de décision qui est connu pour ces mauvaises performances lorsqu'il s'agit d'un large jeu de données avec multiples attributs à cause du problème de sur-apprentissage.

Le travail de Farhan et al. a montré un taux de détection de 80%, un taux de faux négatifs de 27.39% et un taux de faux positifs de 12.81%. Ceci est principalement dû à la diversité introduite dans le jeu de données en ajoutant plus de 50% de flux Botnets inconnus dans la phase de test. En plus, les attributs choisis ne sont pas suffisants pour représenter le flux Botnet dans le SDN ce qui explique le taux de faux négatifs élevé et le taux de détection qui ne dépasse pas 80%.

Tang et al. ont choisi les six attributs les plus basiques pour un système de détection d'intrusion et ils ne se sont pas concentrés sur un type d'attaque particulier. Les performances résultantes sont acceptables mais ne dépassent pas une exactitude de 75.75% et une précision de 83%, ce qui peut mettre le système en danger des attaques Botnets inconnues et limite la sécurité du réseau.

Les performances du travail d'Aboubakr et al. ont montré une excellente exactitude dépassant 97%, une précision de 94.10%, un taux de faux négatifs de 0.71% et un excellent taux de faux positifs avec un taux de 3.81% mais les six attributs choisis ne sont pas spécifiques aux attaques Botnets ce qui peut provoquer une mauvaise détection du trafic malin inconnu de ce type d'attaque.

## 5.5 Conclusion :

Dans ce chapitre, nous avons montré les différentes étapes de simulation des attaques sur l'outil de simulation Mininet et nous avons aussi présenté les différents outils et les différentes caractéristiques de l'environnement matériel utilisé.

Les différents résultats obtenus à partir de l'implémentation de notre solution, en comparaison avec d'autres algorithmes de machine Learning ainsi qu'avec des travaux antérieurs, donnent des conclusions assez satisfaisantes et prometteuses.

# Conclusion Générale

Présenté comme une des récentes révolutions dans le domaine des réseaux informatiques, le réseau SDN présente des avantages non négligeables quant à sa flexibilité et son adaptabilité par rapport aux besoins des nouvelles tendances réseaux et IoT. Cependant, étant donné sa courte durée de vie, cette nouvelle technologie présente encore des brèches de sécurité pas tout à fait résolue. C'est ce que nous avons présenté dans le premier chapitre dans lequel nous avons aussi montré les différentes caractéristiques des réseaux SDNs.

Nous nous sommes ensuite intéressés à une menace particulière qui touche ces réseaux et qui sont les attaques Botnet. Après une présentation de cette attaque et une recherche approfondie sur les différents systèmes de détection d'intrusion de cette dernière, nous avons constaté que l'approche de l'apprentissage profond semble très intéressante pour remédier à notre problème. Pour cette raison, nous avons consacré une partie du deuxième chapitre pour étudier et présenter l'apprentissage automatique et sa branche appelée l'apprentissage en profondeur.

En se basant sur cette technique d'apprentissage en profondeur, nous avons proposé notre solution de détection des attaques de Botnet pour les réseaux SDNs. La solution proposée est constituée de trois modules : le premier module permet la collecte des différents flux installés dans les tables de flux des commutateurs, ces flux permettront au deuxième module d'extraire et de calculer les caractéristiques nécessaires pour la détection des attaques Botnet ; ces caractéristiques vont ensuite permettre au dernier module de classer les flux et déterminer les attaques des flux normaux.

Nous avons ensuite testé notre solution sur un dataset spécialisé aux attaques Botnet en effectuant une sélection des attributs pour subvenir à notre cas qui concerne les réseaux SDNs. Les résultats obtenus ayant été concluant nous avons exposé notre système à une simulation d'attaques spécifiques de Botnet grâce à l'outil Mininet.

Après une comparaison de notre solution avec d'autres algorithmes de machine Learning ainsi qu'avec d'autres travaux antérieurs, notre solution a montré de meilleurs résultats et ainsi nous pensons avoir atteint l'objectif de notre projet de fin d'étude en apportant trois contributions majeures :

- Une architecture détaillée de notre solution.
- Une validation sur le dataset Hogzilla avec une sélection des attributs.
- Une simulation sur Mininet avec un taux de détection assez satisfaisant.

Comme perspective de développement, nous pensons qu'il est possible d'améliorer l'efficacité de notre système avec un dataset plus approprié. En effet, un dataset spécialisé contenant des flux de trafic Botnet dans les SDNs serait encore plus intéressant pour notre cas ; cela permettrait d'avoir de meilleurs résultats lors de la phase d'apprentissage et donc une meilleure détection. De plus, il est intéressant d'effectuer des tests avec d'autres types de réseaux de neurones, comme un réseau d'apprentissage non supervisé, ce qui pourrait nous donner de meilleurs résultats. Enfin, l'ajout d'un module de mitigation à notre système permettra d'agir spontanément dès la détection d'une attaque et de prendre des mesures concrètes pour protéger le réseau SDN.

# Bibliographie et webographie :

- [1] Benzekki, Kamal ; El Fergougui, Abdeslam ; Elbelrhiti Elalaoui, Abdelbaki (2016). "Software-defined networking (SDN) : A survey". Security and Communication Networks. 9(18) : 5803–5833
- [2] Paul Göransson, Chuck Black "Software Defined Networks A Comprehensive Approach" 2014, ISBN : 978-0-12-416675-2
- [3] "SDN Architecture Overview" (PDF). Opennetworking.org. Retrieved 22 November2014. Consulté le 03/09/2020.
- [4] "SDN Controllers Report – SdxCentral" www.sdxcentral.com. Edition 2015. Consulté le 03/09/2020.
- [5] Stallings, William (2015). “Foundations of Modern Networking : SDN, NFV, QoE, IoT, and Cloud” pp 151-153. ISBN-13 : 978-0134175393
- [6] « OpenFlow Table Type Patterns » (PDF). ONF Version No. 1.0 15 August 2014. Consulté le 03/09/2020.
- [7] Wainwright, P., & Kettani, H. (2019). An analysis of Botnet models. Proceedings of the International Conference on Compute and Data Analysis.
- [8] Anagnostopoulos, Marios, Kambourakis, Georgios, Meng, Weizhi, Zhou, Peng. (2019). Botnets : Architectures, Countermeasures, and Challenges. ISBN : 9780367191542.
- [9] Emmanuel C. Ogu, Olusegun A. Ojesanmi, Oludele Awodele and Shade Kuyoro. (2019). A Botnets Circumspection: The Current Threat Landscape, and What We Know So Far.
- [10] Navdeep Kaur, Maninder Singh. « Botnet and Botnet Detection Techniques in Cyber realm ». 2016 International Conference on Inventive Computation Technologies (ICICT).
- [11] Burkov Andriy. (2019). The hundred-page machine Learning book. ISBN : 9781999579517.
- [12] Jason Brownlee. (2016). Master Machine Learning Algorithms - Discover how they work.
- [13] Sebastian Raschka, Vahid Mirjalili. Python Machine Learning : Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow. ISBN : 1787125939.

- [14] <https://tools.ietf.org/html/rfc1459>. Consulté le 03/09/2020.
- [15] <https://mrmint.fr/logistic-regression-machine-Learning-introduction-simple>. Consulté le 03/09/2020.
- [16] Freund, Y. ; Schapire, R. E. (1999). "Large margin classification using the perceptron algorithm".
- [17] C. Livadas, R. Walsh, D. Lapsley and W. T. Strayer, "Using machine Learning techniques to identify Botnet traffic". In Local Computer Networks, Proceedings 2006 31st IEEE Conference on, IEEE, (2006), pp. 967-974.
- [18] E. B. Beigi, H. H. Jazi, N. Stakhanova and A. A. Ghorbani, Towards effective feature selection in machine Learning-based Botnet detection approaches. In Communications and Network Security (CNS), 2014 IEEE Conference on IEEE, (2014), pp. 247-255.
- [19] F. Tariq and S. Baig, "Botnet classification using centralized collection of network flow counters in software defined networks". International Journal of Computer Science and Information Security, vol. 14, no. 8, (2016), pp. 1075.
- [20] Ivan Vasilev, Daniel Slater, Gianmario Spacagna, Peter Roelants and Valentino Zocca, Python Deep Learning : Exploring deep Learning techniques, neural network architectures and GANs with PyTorch, Keras and TensorFlow (2019). ISBN : 978-1789348460.
- [21] Charu C. Aggarwal, Neural Networks and Deep Learning : A Textbook (2018). ISBN : 978-3319944623.
- [22] [http://ressources.unisciel.fr/DAEU-biologie/P2/co/P2\\_chap4\\_c03.html](http://ressources.unisciel.fr/DAEU-biologie/P2/co/P2_chap4_c03.html). Consulté le 03/09/2020.
- [23] Chris Albon, Machine Learning with Python Cookbook : Practical Solutions from Preprocessing to Deep Learning (2018). ISBN : 9781491989388.
- [24] Jeff Heaton, Artificial Intelligence for Humans, Volume 3 : Deep Learning and Neural Networks (2015). ISBN : 1505714346.
- [25] Tang, T.A., Mhamdi, L., McLernon, D., Zaidi, S.A.R., Ghogho, M. : Deep Learning approach for network intrusion detection in software defined networking. In : 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), October 2016.
- [26] Aboubakar, A., Prranggono, B., (2017) Machine Learning based intrusion detection system for software defined networks. Seventh International Conference on Emerging Security Technologies (EST) - Machine Learning based intrusion detection system for software defined networks.

- [27] <https://ids-hogzilla.org/dataset/>. Consulté le 03/09/2020.
- [28] <https://github.com/IvanLetteri/Botnet-SDN-ML>. Consulté le 03/09/2020.
- [29] Jason Brownlee. (2017). Deep Learning with python.
- [30] Jason Brownlee. (2016). Machine Learning Mastery with Python : Understand Your Data, Create Accurate Models and Work Projects End-To-End.
- [31] <https://jupyter.org/>. Consulté le 03/09/2020.
- [32] <https://ubuntu.com/about>. Consulté le 03/09/2020.
- [33] <https://ryu-sdn.org/>. Consulté le 03/09/2020.
- [34] <https://github.com/mininet/mininet/wiki/Documentation>. Consulté le 03/09/2020.
- [35] <https://scapy.readthedocs.io/en/latest/introduction.html#about-scapy>. Consulté le 03/09/2020.
- [36] <https://iperf.fr/>. Consulté le 03/09/2020.
- [37] Craig Schiller, Jim Binkley, Gadi Evron, Carsten Willems, Tony Bradley, David Harley, Michael Cross. (2007). Botnets : The Killer Web App. ISBN : 9781597491358
- [38] <https://doc.ubuntu-fr.org/xterm>. Consulté le 03/09/2020.



# Annexes

## Annexe I : Le script utilisé pour créer la topologie Mininet

Chargement des différentes bibliothèques essentielles pour créer la topologie :

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.node import Host
from mininet.node import OVSKernelSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
```

Création d'une topologie virtuelle en reliant les différents hôtes au commutateurs correspondants et ces derniers au contrôleur, le script pour créer et relier les différents composants est comme suit :

```
def my_network():
    net= Mininet(topo=None, build=False, controller=RemoteController)

    #Add Controller
    info( '***Adding Controller\n' )
    poxController1 = net.addController('c0',controller=RemoteController,ip="127.0.0.1",port=6633)

    #Add Switches
    info( '***Adding Switches\n' )
    #switch12=net.addSwitch('switch12')
    switch1=net.addSwitch('s1')
    switch2=net.addSwitch('s2')

    #Add Hosts
    info( '***Adding Hosts\n' )
    host1=net.addHost('h1', ip='10.0.0.2/24')
    host2=net.addHost('h2', ip='10.0.0.3/24')
    host3=net.addHost('h3', ip='10.0.0.4/24')
    host4=net.addHost('h4', ip='10.0.0.5/24')
    host5=net.addHost('h5', ip='10.0.0.6/24')
    server=net.addHost('h6', mac='00:00:10:00:00:01', ip='10.0.0.7/24')

    #Add Links
    info( '***Creating Links\n' )
    net.addLink(host1,switch1)
    net.addLink(host2,switch1)
    net.addLink(host3,switch1)
    net.addLink(host4,switch1)
    net.addLink(host5,switch1)
    net.addLink(switch1,switch2)
    net.addLink(server,switch2)

    net.build()
```

Les fonctionnalités de démarrage et d'arrêt du réseau SDN :

```
    info( '***Starting network\n' )  
    net.start()  
  
    info( '***Entering command prompt\n' )  
    CLI(net)  
  
    info( '***Stopping network\n' )  
    net.stop()  
  
if __name__ == '__main__':  
    setLogLevel( 'info' )  
    my_network()
```

## Annexe II : Le script utilisé pour la phase d'apprentissage et la création du réseau de neurones approfondi

### Phase 1 : sélection des attributs :

Chargement des librairies essentielles pour la sélection des attributs avec le jeu de données Hogzilla :

```
Entrée [1]: # Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# Importing the dataset
dataset = pd.read_csv('my_Dataset.csv')
X = dataset.iloc[:, :-2].values
y = dataset.iloc[:, -2].values
x1 = dataset.iloc[:, :-2]
x2 = dataset.iloc[:, -2]
```

Application l'algorithme de chi-square et préparation d'un tableau **FeatureScores** contenant les attributs ainsi que leurs scores :

```
Entrée [6]: #apply SelectKBest class to extract top 16 best features
bestfeatures = SelectKBest(score_func=chi2, k=16)
fit = bestfeatures.fit(x1,x2)

Entrée [7]: dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x1.columns)

Entrée [8]: #concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score'] #naming the dataframe columns
```

Affichage des meilleurs scores des attributs :

```
Entrée [9]: featureScores
```

Out[9]:

	Specs	Score
0	dst2src_packets_rate	3.414910e+06
1	src2dst_packets_rate	2.509461e+07
2	dst2src_inter_time_std	5.722353e+08
3	dst2src_packets	9.833517e+06
4	src2dst_inter_time_std	8.173090e+08
5	src2dst_inter_time_max	2.922035e+09
6	flow_use_time	1.098802e+09
7	protocol	5.026486e+05
8	src2dst_packets	3.974861e+05
9	flow_duration	8.393772e+09
10	dst2src_inter_time_max	2.460858e+09
11	packets	6.962806e+06
12	src2dst_inter_time_avg	7.101708e+08
13	flow_idle_time	7.445514e+09
14	bytes	8.021752e+09
15	dst2src_inter_time_avg	2.468956e+08

## Phase 2 : La création de réseau de neurones et l'apprentissage :

Chargement des libraires essentielles pour la préparation du jeu de données et création du réseau de neurones :

```
Entrée [1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sys
from tensorflow.keras.models import load_model
```

Chargement du jeu de données HogZilla avec les attributs sélectionnés :

```
Entrée [3]: dataset = pd.read_csv('my_Dataset.csv')
dataset.head()
```

```
Out[3]:
```

	dst2src_packets_rate	src2dst_packets_rate	dst2src_inter_time_std	dst2src_packets	src2dst_inter_time_std	src2dst_inter_time_max	flow_use_time	protocol
0	2	4	205	3	273	600	0	
1	0	1	0	0	1275	2945	0	
2	0	0	8569	4	25337	80014	0	
3	0	0	0	0	2456	6016	0	
4	0	0	0	0	0	0	0	

```
Entrée [4]: # Importing the dataset
X = dataset.iloc[:, :-2].values
y = dataset.iloc[:, -2].values
```

Création d'un réseau de neurones à 3 couches avec les caractéristiques citées chap. 4.3 et l'application de validation croisée à 5-pli :

```
Entrée [*]: # Importing the Keras libraries and packages
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_validate
from keras.models import Sequential
from keras.layers import Dense
```

```
def build_classifier():
    # Initialising the ANN
    classifier = Sequential()
    # Adding the input layer and the first hidden layer
    classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 16))

    # Adding the second hidden layer
    classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))

    # Adding the third hidden layer
    classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))

    # Adding the output layer
    classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier

classifier = KerasClassifier(build_fn = build_classifier, batch_size = 128, epochs = 100)
scoring = ['accuracy', 'precision', 'recall', 'f1']
results = cross_validate(estimator = classifier, X = X, y = y, cv = 5, n_jobs = -1, scoring=scoring)
```

Les résultats obtenus par la validation croisée :

```
Entrée [8]: accuracies = np.mean(results['test_accuracy'])
precision = np.mean(results['test_precision'])
recall = np.mean(results['test_recall'])
f1_score = np.mean(results['test_f1'])
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Precision: {:.2f} %".format(precision.mean()*100))
print("Recall: {:.2f} %".format(recall.mean()*100))
print("f1_score: {:.2f} %".format(precision.mean()*100))

('Accuracy:', 0.9747394533436191)
('Precision:', 0.9683412294031764)
('Recall:', 0.9816498519905758)
('F1_score:', 0.9749501252493739)
```

La matrice de confusion :

```
Entrée [7]: # Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
Out[7]: array([[63885, 2125],
               [ 1215, 64997]])
```

Sauvegarde du modèle qui sera utiliser dans la phase de classification :

```
Entrée [25]: # save model and architecture to single file
classifier.save("3Final.h5")
print("Saved model to disk")
```

```
Saved model to disk
```