

**UNIVERSITÉ DES SCIENCES ET DE LA
TECHNOLOGIE HOUARI BOUMEDIENE**

MÉMOIRE DE MASTER

**Une approche de clustering pour la
détection des attaques DoS dans un
réseau SDN**

Auteurs :

DRIA SALIM
BENCHIKH ELHOCINE MED
REDA

Superviseur :

Pr. Boughaci

Membre du jury :

BENCHIKH MED REDA
BENCHIKH MED REDA

*Mémoire présentée en vue de l'obtention du diplôme de maîtrise en Réseaux et
Systèmes Distribués*

29 juillet 2020

Table des matières

Liste des abréviations	v
Introduction générale	vi
1 Software Defined Networking	1
1.1 Limites des réseaux traditionnelles	1
1.2 Une approche réseau moderne	2
1.3 Définition	2
1.4 Architecture SDN	2
1.4.1 Plan de Données	4
1.4.2 Plan de Contrôle	4
1.4.3 Plan d'Applications	5
1.5 Caractéristiques des SDN	5
1.6 OpenFlow et SDN	5
1.6.1 Table de Flux	6
1.7 Contrôleur SDN	7
1.8 Avantages et inconvénients des SDN	7
1.8.1 Avantages	7
1.8.2 Inconvénients	7
1.9 Conclusion	8
2 Sécurité des réseaux	9
2.1 Sécurité dans un réseau Informatique	9
2.1.1 Généralités	9
2.1.2 Les menaces d'un réseau informatique	10
2.1.3 Les mécanismes de sécurité	10
2.2 La sécurité dans les réseaux SDN	10
2.2.1 Les failles de sécurité des réseaux SDN	11
2.2.2 Attaques ciblées sur les couches SDN	12
2.3 Les attaques DoS	13
2.3.1 Définition	13
2.3.2 Fonctionnement	13
2.3.3 Classification d'une attaque DoS	14
2.3.4 Impacts des attaques DoS sur un réseau SDN	15
2.4 Système de Détection d'Intrusions (IDS)	16
2.4.1 Définition	16
2.4.2 Les approches	17
2.4.3 Les méthodes de détection d'intrusions	17
2.4.4 Les méthodes de collection d'informations sur les flux	17
2.4.5 Évaluation des performances d'un NIDS	18

2.5	Travaux existants	19
2.5.1	Flow based Intrusion Detection System [11] :	20
2.5.2	Machine based Intrusion Detection System [13] :	21
2.5.3	Comparaison entre ces deux IDS	22
2.6	Conclusion	22
3	Data Mining	23
3.1	Data Minig	23
3.1.1	Définition	23
3.1.2	Ètapes du processus KDD	23
3.1.3	Dataset	24
3.2	Apprentissage automatique	25
3.2.1	Apprentissage supervisé	25
3.2.2	Apprentissage non supervisé	26
3.2.3	Conclusion	28
4	Conception	29
4.1	Problématique	29
4.2	Motivation du travail	30
4.3	Présentation de la Solution	31
4.3.1	Module MEI	32
4.3.2	Module F-Clustering	33
4.4	Conception du module MEI	33
4.5	Conception du module F-Clustering	34
4.5.1	Préparation des données	34
4.5.2	Recherche du modèle	36
4.6	Module d'archivage	37
4.7	Conclusion	37
5	Implémentation, simulation et résultats	38
5.1	Environnement de travail	38
5.1.1	Contrôleur SDN	38
5.1.2	Simulation du réseau SDN	39
5.2	Langage et Librairies utilisées	40
5.2.1	Python	40
5.2.2	Scikit-learn	40
5.2.3	Pandas	40
5.2.4	Argus	41
5.3	Réalisation du module F-Clustering	41
5.4	Évaluation et test des performances	42
5.5	Simulation	44
5.5.1	Création de la topologie	44
5.5.2	Installation des fonctions	45
5.5.3	Simulation des attaques	50
5.6	Conclusion	52

Table des figures

1.1	Plan de données et plan de contrôle	3
1.2	Architecture SDN	3
2.1	Fonctionnement des systèmes de détection d'intrusions	16
2.2	IDS [11]	20
3.1	K-Means	28
4.1	Principe de dénie de service réfléctif	30
4.2	Architecture et fonctionnement de F-DoS	32
4.3	Architecture du module MEI	34
4.4	Étapes de conception de F-Clustering	37
5.1	Comparaison entre les résultats obtenus selon l'opération de prétraitement effectuée	43
5.2	Architecture de notre réseau SDN	44
5.3	Création de la topologie sous mininet	45
5.4	Activation du Port-Mirroring	46
5.5	Installation du module de détection	46
5.6	Installation du serveur d'applications	47
5.7	Installation du serveur de fichiers	47
5.8	Établissement de connexion entre le client et le serveur d'applications	48
5.9	Capture et analyse du trafic UDP	48
5.10	Établissement de connexion entre le client et le serveur de fichiers	49
5.11	Simulation d'une attaque RDoS	50
5.12	Détection de l'attaque RDoS	51
5.13	Simulation d'une attaque UDP-Flooding	51
5.14	Détection de l'attaque UDP-Flooding	52

Liste des tableaux

1.1	Entrée d'une table de flux	6
1.2	Match Fields	6
2.1	Vecteurs d'attaque	11
2.2	Matrice de confusion	18
2.3	Comparaison entre les deux IDS proposés	21
3.1	Quelques Datasets disponibles pour les système de détection d'intrusions	25
4.1	Déscription des attributs	36
5.1	Matrice de confusion	42
5.2	Mesures de performance	42
5.3	Impact des prétraitements sur les performances	43
5.4	Rôle de chaque hôte dans la topologie mininet	45

Liste des abréviations

SDN	Software Defined Networking
API	Application Programming Interface
ONF	Open Networking Foundation
OF	Open Flow
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
DoS	Denial Of Service attack
RDoS	Reflective Denial Of Service attack
NIDS	Network Intrusion Detection System
HIDS	Host Intrusion Detection System
SVM	Support Vector Machine
KDD	Knowledge Discovery in Database
MEI	Module d'Extraction des Informations

Introduction générale

Durant ces deux dernières décennies les technologies de l'information ont énormément évolué et leur domaine d'application est de plus en plus vaste. L'internet est utilisée dans tous les secteurs, industrie, éducation, commerce,...etc. La demande des services Cloud augmente considérablement et les gens utilisent de plus en plus les réseaux sociaux. Malheureusement les réseaux traditionnels ne sont pas adéquat à ces nouvelles technologies de l'information pour plusieurs raisons, on peut citer ; leur scalabilité limitée, ils consomment trop de ressources matériels, leur structure est complexe, ils sont aussi coûteux et difficiles à gérer. Ce qui oblige à concevoir des réseaux évolutifs, fiables et robustes pour faire face à ces problèmes.

Comme solution la technologie Software Defined Networking a été proposée, son but est de rendre les réseaux programmables par logiciel contrairement aux réseaux traditionnels qui nécessitaient l'intervention humaine pour configurer chaque équipement à part qui est une opération fastidieuse, prend beaucoup de temps et exposé aux erreurs de configuration. Le SDN rend ce concept de réseau programmable possible en introduisant un nouvel équipement dit contrôleur, ce dernier va être le cerveau du réseau, toute décision liée au routage du trafic réseau, la sécurité, la gestion des équipements est prise par le contrôleur.

Cependant cette technologie souffre de nombreux problèmes de sécurité certains sont hérités de l'environnement réseau, tandis que d'autre sont propre à l'architecture SDN. La nature centralisée de l'architecture SDN est exploitée pour mener des attaques contre le contrôleur, une fois compromis, l'attaquant aura un contrôle total du réseau, il peut reconfigurer les équipements, rediriger le trafic réseau ailleurs, lancer d'autre attaques. Donc il est essentiel de bien sécuriser l'architecture SDN, spécialement le contrôleur, pour garantir le bon fonctionnement du réseau.

Le travail qui nous a été confié est de concevoir un système de détection des attaques par déni de service (DoS) dans un réseau SDN en utilisant l'apprentissage automatique, plus précisément le Clustering.

Afin de bien mener le travail dans le cadre approprié, ce mémoire est structuré comme suit ; Dans un premier lieu, nous commençons par définir le concept des réseaux programmables par logiciel, voir les avantages et les inconvénients qu'apporte cette nouvelle technologie, on passera par la suite à la sécurité des réseaux SDN où on étudiera en détail l'aspect sécurité des réseaux, on verra les majeures contraintes de sécurité qui doivent être respectées pour garantir le bon fonctionnement du SDN, les différents type d'attaques auxquels un environnement SDN est exposé et une étude comparative entre des travaux existants dans le même contexte que notre travail. Nous entamerons ensuite la conception en commençant par l'analyse et la spécification des besoins pour passer à

la modélisation du système en décrivant son architecture générale et les différents modules qui le composent. Une fois notre système réalisé, il ne reste que le mettre sous le test pour évaluer ces performances et son efficacité dans un réseau SDN qu'on va simuler. La dernière section de ce mémoire sera une conclusion générale sur le travail qu'on a fait.

Chapitre 1

Software Defined Networking

Les nouvelles exigences des réseaux modernes en terme d'adaptabilité, d'automatisation, de maintenabilité ont fait naître une nouvelle technologie de réseaux informatique, connue sous le nom de *Software Defined Networking*, qui donne un contrôle centralisé et programmable des ressources réseaux.

Ce chapitre donne un aperçu sur l'environnement SDN et montre comment il est conçu pour répondre aux besoins changeants du réseau.

1.1 Limites des réseaux traditionnelles

Avant de passer au Software-Defined Networking, rappelons deux majeures limites des réseaux traditionnels cités par l'ONF (Open Networking Foundation) [1] :

- **Architecture statique et complexe** : pour répondre aux différents niveaux de qualité de service, le volume de trafic élevé et les exigences en matière de sécurité ; la technologie réseau est devenue plus complexe et difficile à gérer. Cela a donné lieu à un certain nombre de protocoles définis de façon indépendante dont chacun répond à une partie des besoins de réseautage. Un exemple de la difficulté que cela présente est lorsque les appareils sont ajoutés ou déplacés. Le personnel de gestion du réseau doit utiliser des outils de gestion au niveau de l'appareil pour modifications des paramètres de configuration dans plusieurs commutateurs, routeurs, pare-feu,...etc. Les opérations de mise à jour sont fastidieuses et coûteuses en matière de temps et donnent place aux erreurs de configuration qui peuvent engendrer des problèmes au niveau du réseau.
- **Scalabilité limitée** : la demande sur les réseaux augmente rapidement, tant en volume qu'en variétés. Ajouter plus de commutateurs ou augmenter la capacité de transmission est difficile en raison de la complexité et nature statique du réseau.

1.2 Une approche réseau moderne

Une approche réseau moderne doit satisfaire les exigences citées par L'Open Data Center Alliance (ODCA) [2], notamment :

- **Adaptabilité** : les réseaux doivent s'adapter dynamiquement, en fonction des besoins des applications, des activités, la politique et les conditions du réseau.
- **Automatisation** : les changements de politique doivent être propagé automatiquement afin de réduire le travail manuel et les erreurs.
- **Sécurité intégrée** : les applications réseau doivent intégrer la sécurité comme service de base et non pas comme solution complémentaire.
- **Mise à l'échelle sur demande** : les implémentations doivent avoir la capacité à prendre de l'expansion ou à réduire le réseau et ses services pour répondre aux requêtes à la demande.

1.3 Définition

Software-defined networking est une nouvelle architecture réseau, qui a pour but pratique de rendre programmables les réseaux par le biais d'un contrôleur centralisé. Rappelons que dans les réseaux traditionnels les périphériques réseaux (commutateurs et routeurs) construisent leurs tables de routage localement, ce qui signifie qu'ils prennent leurs propres décisions en interne sur la meilleure façon d'aiguiller le trafic.

Dans les SDN les décisions de routage sont prises par un contrôleur centralisé ce qui fait que les périphériques n'ont plus intérêt à prendre des décisions localement, ils n'ont qu'à suivre celles prises par le contrôleur. Ce qui rend cette architecture dynamique, programmable et idéale pour les réseaux modernes à haute demande.

1.4 Architecture SDN

Le concept central derrière l'architecture SDN est de permettre aux développeurs et gestionnaires de réseau d'avoir un contrôle centralisé sur les équipements réseau.

Le SDN fait la séparation entre le plan de contrôle et le plan de données. Le plan de contrôle est responsable des décisions de transmission, il comprend des mécanismes de transmission d'itinéraire du trafic. Le plan de données représente la partie des commutateurs et routeurs qui assurent effectivement la transmission des données. La différence entre l'architecture traditionnelle et l'architecture SDN est illustrée dans la figure suivante :

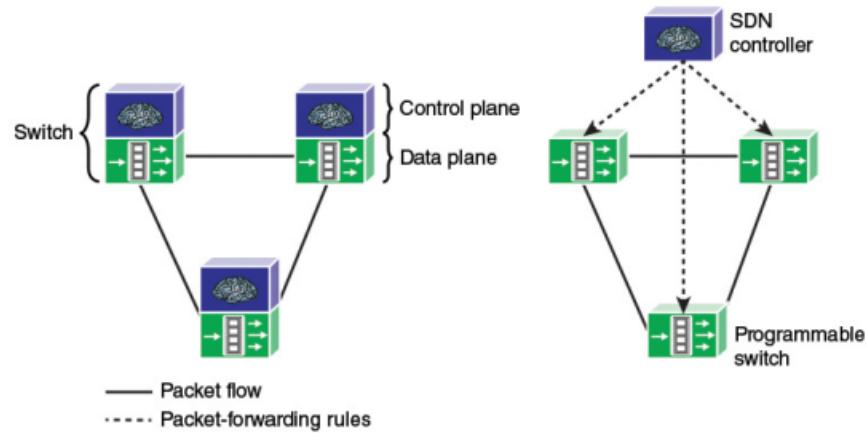


FIGURE 1.1 – Plan de données et plan de contrôle

Dans l’architecture traditionnelle le plan de données et le plan de contrôle sont intégrés dans un même équipement physique. Par contre dans Le SDN le plan de contrôle est externalisé de tous les périphériques réseaux et associé à un équipement dit contrôleur.

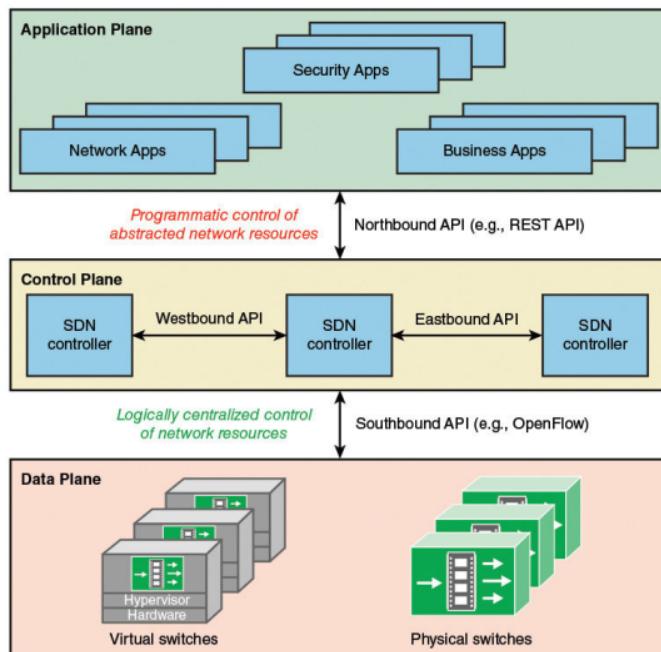


FIGURE 1.2 – Architecture SDN

L’architecture SDN définit trois couches, comme indiqué dans la figure ??, chacune représente un plan, plan de données, plan de contrôle ou plan d’application. Chaque couche ne peut communiquer qu’avec ces couches voisines.

1.4.1 Plan de Données

Le plan de données se compose de commutateurs physiques et de commutateurs virtuels, qui sont responsables de la transmission des paquets. Cependant chaque commutateur doit implémenter un modèle de transfert de paquet, ce modèle est défini à travers une API (Application programming interface) qui réside entre le plan de contrôle et le plan de données permettant la communication entre le contrôleur et les commutateurs du plan de données. Parmi ces API on peut citer OpenFlow, qui sera détaillé par la suite dans ce chapitre.

Le plan de données assure une fonction autre que la transmission des données, le support de contrôle. Cette fonction permet la communication avec le contrôleur SDN pour l'échange des informations liées à la gestion du périphérique.

1.4.2 Plan de Contrôle

Le plan de contrôle SDN, composé d'un ou plusieurs contrôleurs, mappe les demandes de service de couche d'application en commandes et directives spécifiques aux commutateurs du plan de données et fournit aux applications des informations sur la topologie et l'activité du plan de données.

Le contrôleur définit les flux de données qui se produisent dans le plan de données. Un flux est une séquence de paquets qui partagent un ensemble de valeurs de champ d'en-tête. Pour qu'un flux traverse le réseau, le contrôleur intervient pour vérifier si ce flux est autorisé dans le cadre de la politique réseau définie.

Dans ce qui suit, deux frameworks les plus utilisés pour la mise en œuvre des contrôleurs SDN :

OpenDaylight[3] : Un framework open source, écrit en java, permettant la programmation réseau. OpenDaylight a été développé par Cisco et IBM, il permet à plusieurs contrôleurs distribués, résidants dans des serveurs différents, de fonctionner comme étant un seul contrôleur centralisé.

Ryu[4] : Framework open source développé par NTT labs, basé composant et entièrement écrit en python. Ryu fournit des composants logiciels avec API bien définie qui rend la tâche facile aux développeurs pour créer de nouvelles applications de gestion et de contrôle.

NOX[5] : Développé par Nicira networks, NOX est une plateforme conçue spécialement pour le développement des applications de contrôle de réseau.

1.4.3 Plan d'Applications

Le plan d'application contient des applications et des services qui définissent, surveillent et contrôlent les ressources et le comportement du réseau. Ces applications interagissent avec le plan de contrôle via des interfaces dédiées pour que la couche de contrôle personnalise automatiquement le comportement et les propriétés des ressources réseau. La programmation d'une application SDN utilise la vue abstraite des ressources réseau fournie par la couche de contrôle au moyen d'informations et de modèles de données exposés via l'API fournie.

1.5 Caractéristiques des SDN

Après avoir vu la nouvelle architecture réseau, le software-defined Networking, en opposition avec l'architecture traditionnelle, on peut dire que les principales caractéristiques des SDN sont les suivantes :

- **Séparation des plans** : le plan de contrôle est séparé du plan de données. Les périphériques de plan de données deviennent des simples dispositifs de transfert de paquets. Le plan de contrôle est installé dans un contrôleur ou ensemble de contrôleurs centralisés et coordonnés.
- **Programmabilité** : le réseau est programmable par des applications qui s'exécutent au-dessus du contrôleur SDN. Les contrôleurs présentent une vue abstraite des ressources du réseau aux applications.
- **Centralisation** : le cerveau du réseau est (logiquement) centralisée dans des contrôleurs SDN basé sur des logiciels qui maintiennent une vue globale du réseau.

1.6 OpenFlow et SDN

Pour implémenter le concept SDN en pratique, un protocole standardisé et sécurisé est essentiel pour établir la connexion entre le contrôleur et les ressources réseau. L'architecture Openflow se compose de trois éléments principaux : un commutateur Openflow, un contrôleur externe et le protocole Openflow. Le plan de données et le plan de contrôle communiquent sur un canal sécurisé via ce protocole. Le commutateur Openflow dispose de tables de flux et d'une couche d'abstraction qui communique en toute sécurité avec un contrôleur via le protocole Openflow. La table de flux contient des entrées de flux qui déterminent comment les paquets appartenant à un flux seront traités et transmis. Les entrées de flux consistent en :

- Matching rules – pour apparier les paquets entrants.
- Counters – pour collecter des statistiques du flux.
- Set of instructions / actions - à appliquer en cas de correspondance.

Lorsqu'un paquet arrive à un commutateur Open Flow, les champs d'en-tête du paquet sont extraits et comparés aux règles correspondances. Si une correspondance

est trouvée, le commutateur applique l'action appropriée. S'il n'y a pas de correspondance, l'action effectuée par le commutateur dépend de l'instruction définie par l'entrée de *table-miss flow*. Chaque table de flux doit avoir cette entrée afin de gérer la non-correspondance d'un flux dans une table. Des exemples d'actions qui peuvent être effectuées lorsque aucune correspondance n'est trouvée sont : abandonner le paquet, continuer le processus de correspondance sur la table de flux suivant, acheminer le paquet au contrôleur.

1.6.1 Table de Flux

La table de flux contient plusieurs entrées. Chaque entrée de cette table a la structure suivante :

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie
--------------	----------	----------	--------------	----------	--------

TABLE 1.1 – Entrée d'une table de flux

- **Match Fields** : permet de sélectionner des paquets qui correspondent aux valeurs des champs de Match Fields.
- **Priority** : un champ sur 16 bits, définit la priorité de l'entrée. la valeur 0 correspond à la priorité la plus basse.
- **Counters** : un conteur de paquets. Il est mis à jour lors d'une correspondance d'un paquet à une entrée de la table de flux.
- **Instructions** : ensemble d'instructions à déclencher en cas de correspondance.
- **Timeouts** : durée de vie d'une entrée de table.
- **Cookie** : utilisé par le contrôleur pour le filtrage de paquets.

Le champ principal dans une entrée de table de flux est le **Match Fields**, ce dernier définit un vecteur d'attributs correspondant aux champs de l'en-tête du paquet arrivé au commutateur.

Port Ent	Port Srt	Ethr AS	Ethr AD	Ethr Type	IP	AS IPv4	AD IPv4	AS IPv6	AD IPv6	TCP Src	TCP Dest	UDP Src	UDP Dest
----------	----------	---------	---------	-----------	----	---------	---------	---------	---------	---------	----------	---------	----------

TABLE 1.2 – Match Fields

Port Ent : Identificateur du port par lequel le paquet est arrivé.

Port Srt : Identificateur du port de sortie.

Ethr AS et Ethr AD : Adresses Ethernet, source et destination.

Ethr Type : Type de trame Ethernet.

IP Type : version IP, 4 ou 6

AS IPv4 et AD IPv4 : Adresses IPv4, source et destination.

AS IPv6 et AD IPv6 : Adresses TPv6, source et destination.

TCP Src et TCP Dest, UDP Src et UDP Dest : Ports source et destination de couche de transport.

1.7 Contrôleur SDN

Comme mentionné précédemment, le contrôleur est le composant principal de toute l'infrastructure SDN où tous les calculs y sont effectués. Il est responsable de la gestion de tout le réseau.

En résumé, les fonctions assurées par un contrôleur SDN sont [6] :

- **Routage plus court chemin** : utilise les informations de routage collectées à partir des commutateurs pour établir la table de routage.
- **Gestion de notifications** : reçoit, traite et transmet à un événement d'application, les notifications d'alarme, les alarmes de sécurité et les changements d'état.
- **Sécurité** : fournit des services de sécurité.
- **Gestion de topologie** : construit et maintient les informations sur l'interconnexion des équipements réseau.
- **Gestion de statistique** : collecte et stock des informations sur le trafic réseau.
- **Gestion de périphériques** : configure les paramètres des commutateurs et gère les tables de flux [section 1.6](#).

1.8 Avantages et inconvénients des SDN

1.8.1 Avantages

- Économies opérationnelles : les SDN réduisent les dépenses d'exploitation. Les services de réseau peuvent être gérés par des applications, libérant ainsi l'équipe de réseautage.
- Une meilleure gestion : les SDN offrent une gestion centralisée et automatisée des équipements réseau.
- Flexibilité : les SDN créent une flexibilité dans la façon dont le réseau peut être exploité. Les administrateurs réseau peuvent écrire leurs propres services de réseau en utilisant les outils de développement standard.
- Disponibilité améliorée : en éliminant l'intervention manuelle, les SDN permettent de réduire les erreurs de configuration et de déploiement qui peuvent affecter le réseau.

1.8.2 Inconvénients

- Il faut modifier entièrement l'infrastructure réseau pour mettre en œuvre le SDN. Il faut donc reconfigurer complètement le réseau. Cela implique des dépenses considérables en vue de mettre en œuvre cette technologie.
- SDN à ses propres vulnérabilités de sécurité. Ceci nécessite de mettre en place de nouvelles approches pour veiller sur la sécurité du réseau.
- Toute l'intelligence du réseau est placée dans le contrôleur, ce qui constitue un point de défaillance unique. Si le contrôleur tombe en panne ou il est compromis tout le réseau cessera de fonctionner.

1.9 Conclusion

Le Software Defined Networking annonce des changements importants sur les réseaux dans les années à venir. Ce chapitre n'a parlé que brièvement sur cette nouvelle technologie, sa définition est bien plus vague et son domaine d'application est bien plus large. Les profits qui peuvent être tirés de cette technologie sont innombrables si bien exploitée.

Chapitre 2

Sécurité des réseaux

Les réseaux définis par logiciel sont une catégorie émergente de réseaux qui présente de nouveaux défis du point de vue sécurité. Ce chapitre fournit un aperçu complet sur les menaces de sécurité auxquelles est confrontée cette architecture émergente, en particulier les attaques par déni de service Dos. Différentes solutions déjà proposées pour sécuriser cette technologie vont être vues et examiner au cours de ce chapitre afin de mieux comprendre comment l'aspect de sécurité est géré au sein d'un réseau SDN, ce qui nous aidera par la suite à proposer une solution adéquate pour la détection des attaques DoS dans un tel réseau.

2.1 Sécurité dans un réseau Informatique

2.1.1 Généralités

La sécurité des réseaux informatiques est la stratégie et les dispositions d'une organisation pour assurer la sécurité de ses actifs et de tout le trafic réseau. Elle se manifeste par une mise en œuvre des mécanismes qui sont conçues pour détecter, prévenir et lutter contre une attaque de sécurité et des services de sécurité qui augmentent la sécurité des traitements et des échanges de données d'un système. Un service de sécurité utilise un ou plusieurs mécanismes de sécurité.

Lors du développement d'un réseau sécurisé, il convient de prendre en compte les services suivants :

- **L'authenticité** : L'identité des acteurs de la communication est vérifiée.
- **La confidentialité** : Les données (et l'objet et les acteurs) de la communication ne peuvent pas être connues d'un tiers non autorisé.
- **L'intégrité** : Les données de la communication n'ont pas été altérées.
- **La disponibilité** : Les acteurs de la communication accèdent aux données dans de bonnes conditions.
- **La non-réputation** : Les acteurs impliqués dans la communication ne peuvent nier y avoir participé.

2.1.2 Les menaces d'un réseau informatique

Une attaque est une action qui compromet la sécurité des informations. Les attaques peuvent être classées sous deux catégories :

Attaque passive : consiste à écouter sans modifier les données ou le fonctionnement du réseau. Elles sont généralement indétectables mais une prévention est possible.

Attaque active : consiste à modifier des données ou des messages, à s'introduire dans des équipements réseau ou à perturber le bon fonctionnement de ce réseau. Noter qu'une attaque active peut être exécutée sans la capacité d'écoute.

2.1.3 Les mécanismes de sécurité

- **Pare-feu** : un élément (logiciel ou matériel) du réseau informatique contrôlant les communications qui le traversent. Il a pour fonction de faire respecter la politique de sécurité du réseau, celle-ci définissant quels sont les communications autorisés ou interdits. N'empêche pas un attaquant d'utiliser une connexion autorisée pour attaquer le système. Ne protège pas contre une attaque venant du réseau intérieur (qui ne le traverse pas).
- **Antivirus** : logiciel sensé de protéger ordinateur contre les logiciels (ou fichiers potentiellement exécutables) néfastes. Ne protège pas contre un intrus qui emploie un logiciel légitime, ou contre un utilisateur légitime qui accède à une ressource alors qu'il n'est pas autorisé à le faire.
- **Détection d'intrusion** : repère les activités anormales ou suspectes sur le réseau surveillé. Ne détecte pas les accès incorrects mais autorisés par un utilisateur légitime. Mauvaise détection : taux de faux positifs, faux négatifs.
- **Contrôle d'accès** : vérifie les droits d'accès d'un acteur aux données. N'empêche pas l'exploitation d'une vulnérabilité.

2.2 La sécurité dans les réseaux SDN

Le but des SDN est de rendre les réseaux traditionnels plus sécurisés et robustes. La gestion centralisée des SDN simplifie la façon dont les mécanismes de sécurité sont introduits dans le réseau. Cependant cette solution ouvre porte à d'autres types d'attaque. Attaques initiées à partir des applications de gestion malicieuses, attaques sur le contrôleur, compromettre les switches.

Selon Kreutz et al[7], les vecteurs d'attaques identifiés dans les SDN sont décrits dans le tableau 2.1.

N.	Vecteur d'attaque
1	Flux de trafic falsifié ou truqué
2	Attaques contre les vulnérabilités des commutateurs
3	Attaque sur les liens de communication du contrôleur
4	Attaques contre les vulnérabilités des contrôleurs
5	Manque de mécanismes pour assurer la confiance entre le contrôleur et application de gestion
6	Attaques contre les vulnérabilités des postes administratifs

TABLE 2.1 – Vecteurs d'attaque

Parmi ces vecteurs d'attaques, numéro 3,4 et 5 ne sont pas présents dans les réseaux traditionnels. Ils sont spécifiques aux SDN. Ils résultent de la séparation du plan de contrôle et de données et de la centralisation logique des contrôleurs. D'autres vecteurs étaient déjà présentés dans les réseaux traditionnels.

2.2.1 Les failles de sécurité des réseaux SDN

Du point de vue sécurité, les différences majeures du modèle SDN avec les installations traditionnelles sont :

- La centralisation du contrôle. Le contrôleur est un point critique du réseau (d'autant plus si aucun contrôleur auxiliaire n'a été déployé), et il doit être l'unique source des consignes « OpenFlow » envoyées aux « switches ».
- L'accès programmatique explicite offert aux clients, qui sont généralement des entités organisationnelles ou commerciales distinctes, ce qui présente des exigences qui n'existent pas dans les domaines administratifs fermés, notamment en ce qui concerne la protection de l'intégrité du système, les données tierces et les interfaces ouvertes.
- Les interfaces et protocoles SDN sont en cours de développement, ce que peut conduire à des problèmes de sécurité. Toutefois, la réintégration des fonctionnalités de sécurité des technologies existantes et la compatibilité des protocoles existants ne sont pas évidentes.
- La connexion entre domaines est une exigence qui nécessite la possibilité de connecter une infrastructure de différents domaines. Cela peut être réalisé en connectant des contrôleurs de différents fournisseurs via l'I-CPI. Cela nécessite des mécanismes permettant d'établir des relations de confiance, de déterminer le niveau d'autorisation pour prévenir les abus et établir des canaux sécurisés.

2.2.2 Attaques ciblées sur les couches SDN

Dans le contexte de SDN, les attaques sont classées en fonction de la couche cible. Une attaque peut cibler une ou plusieurs couches à la fois.

Couche d'applications :

La plupart des fonctions réseau peuvent être programmées dans des applications SDN, les parties malveillantes peuvent prendre le contrôle du réseau en injectant des applications SDN au niveau de la couche de contrôle. De plus il n'existe pas de normes qui régularisent l'utilisation des API SDN par les applications pour le contrôle du réseau. Par conséquent les applications développées par différents fournisseurs sous différents environnements peuvent causer des problèmes d'interopérabilité, de collision entre les applications et de violation des politiques de sécurité.

Couche de données :

- Avec la séparation des plans de données et de contrôle, les pirates essayent de mettre en erreur le switch en lui envoyant des faux messages de contrôle dans le but de détourner le trafic. Les versions récentes de l'OpenFlow implémentent le protocole TLS qui établit une connexion sécurisée entre le switch et le contrôleur mais son utilisation est optionnelle et ignorée par beaucoup d'usagers.

- Les switches OpenFlow sont dotés de tables de flux avec des tailles limitées. Dans le cas où les flux sont mis en correspondance d'une manière très granulaire les tables des switches risquent d'être saturées.

- Certaines attaques peuvent exploiter la limite des buffers au niveau des switches utilisés pour enregistrer les flux en attendant la réponse du contrôleur par rapport aux règles à appliquer sur le flux en envoyant plusieurs nouveaux flux dans un très petit intervalle de temps dans le but de les saturer.

Couche de contrôle :

Le contrôleur est le point le plus sensible dans un réseau SDN. Pour cette raison il est vu comme une cible privilégiée des attaques. Les menaces liées à cette couche :

- Les applications situées au-dessus du plan de contrôle peuvent causer des dangers au contrôleur. Ce dernier peut servir plusieurs applications simultanément ce qui rend leur authentication et leur attribution d'autorisations une fonction compliquée. Par conséquent certaines applications malveillantes pourraient accéder aux services du contrôleur pour compromettre le réseau.
- La centralisation du plan de contrôle au niveau du contrôleur SDN facilite le contrôle du réseau mais elle peut causer des problèmes. En effet, si le contrôleur est sollicité pour définir les règles pour chaque nouveau flux, il risque d'être saturé dans un réseau à grande échelle. L'ajout de nouveaux contrôleurs pour distribuer les charges entre ces derniers n'est pas la meilleure solution dans ce type

de situation. Si le réseau est assez grand cette solution causera un blocage en cascade.

- Les attaques DoS sont parmi les attaques les plus dangereuses dans les réseaux SDN. L'objectif de ces attaques est de saturer le contrôleur en utilisant des techniques qui provoquent l'envoi d'un grand nombre de paquets OpenFlow vers le contrôleur.

2.3 Les attaques DoS

Parmi les problèmes majeurs de la sécurité des réseaux SDN, les attaques DoS dont le nombre ne cesse pas d'augmenter chaque année et occasionne beaucoup de dégâts. Le but d'une telle attaque n'est pas de récupérer ou d'altérer des données, mais de nuire au fonctionnement de réseau. Elles sont conduites grâce à des outils parasites. La difficulté vient du fait que l'outil utilisé est, chaque fois, renouvelé, et qu'il est donc impossible d'anticiper l'attaque. Le principe général des attaques DoS, consiste à envoyer des données ou des paquets dont la taille ou le contenu est inhabituel, ceci a pour effet de provoquer des réactions inattendues du réseau, pouvant aller jusqu'à l'interruption de service.

2.3.1 Définition

Une attaque DoS (Denial of Service) en français déni de service, vise à rendre indisponibles pendant un temps indéterminé les services ou les ressources d'une organisation, de façon à l'empêcher d'offrir ces services. Cette attaque peut ainsi bloquer un serveur de fichiers, rendre impossible l'accès à un serveur web ou empêcher la distribution de courriel dans une entreprise. Les victimes du déni de service ne sont pas uniquement celles qui le subissent, les postes compromis (daemons et masters) et les postes clients qui n'arrivent pas à accéder aux services désirés sont également des victimes.

2.3.2 Fonctionnement

La réalisation d'un DoS n'est pas très compliquée, mais pas moins efficace, que la plupart de temps, elles sont réalisées avec succès car la plupart des DoS exploitent des failles liées au protocole TCP/IP. Les contre-mesures sont compliquées à mettre en place, d'autant plus que ce type d'attaque utilise les services et protocoles normaux des réseaux. La seule façon de s'en protéger est de détecter les comportements anormaux, ce qui implique notamment la vérification de l'intégrité des paquets, la surveillance du trafic, l'établissement de profils types et de seuils. L'attaque DoS se fait à partir d'une seule machine. Dans ce type d'attaque, le pirate lance seul son attaque contre la victime. La plupart du temps, le pirate cache son identité réseau (adresse IP et ports udp/tcp) en se faisant passer pour une ou plusieurs machines (IP Spoofing). Ainsi, il ne peut pas être reconnu par la victime.

2.3.3 Classification d'une attaque DoS

Les attaques DoS prennent de multiples formes et utilisent de nombreuses méthodes différentes pour mettre hors-service le réseau. Pour rendre le réseau hors service, il existe 3 stratégies :

- Saturer la bande passante.
- Epuiser les ressources système.
- Cibler une faille logicielle particulière.

A) Attaque par complexité d'algorithme :

Une forme d'attaque qui exploite les cas connus dans lesquels un algorithme utilisé dans un logiciel présentera un comportement de complexité correspondant au pire de ses cas. Ce type d'attaque peut être utilisé pour ralentir des serveurs ou des processus ou même les faire planter.

B) Attaque par inondation (Flooding) :

Le principe est de saturer la bande passante d'un réseau, ce qui empêche d'autres clients d'accéder à une ressource ou à un serveur, parmi les attaques de type inondation les plus utilisés actuellement par les pirates, nous citons les attaques SYN Flood, UDP Flood et ICMP Flood.

SYN Flood : cette technique d'attaque s'applique dans le cadre d'un protocole TCP et vise principalement à submerger le serveur cible d'une grande quantité de requêtes SYN (Synchronized).

UDP Flood : cette attaque exploite le mode non connecté du protocole UDP. Elle consiste à générer une grande quantité de paquets UDP destinés à une machine cible (aussi appelée victime).

ICMP Flood : les attaques par inondation ICMP également appelées « smurf attacks ». Ils inondent une victime avec un grand nombre de paquets ICMP utilisant des adresses IP sources usurpées. Le serveur victime répondra au propriétaire sans méfiance de l'adresse IP usurpée avec les réponses de l'ICMP.

C) Attaque par réflexion/amplification

Reflection Denial of Service Amplification Attacks sont des types spécifiques des attaques DoS, où l'attaquant utilise des appareils intermédiaires pour refléter son trafic. Geva, e al [8] on fait la différence entre les attaques d'amplification et les attaques de réflexion, ils déclarent que les attaques par amplification sont plus simples alors que les attaques par réflexion pourraient amener un émetteur ou un récepteur à retransmettre des paquets plusieurs fois, ce qui les rend plus complexes. On pense que cette distinction pourrait prêter à confusion en raison du fait que la plupart des publications examinées

ne les désignent pas comme des types d'attaques distincts. Par conséquent, pour éviter toute confusion, ce travail considérera les attaques d'amplification et de réflexion comme synonymes. Les appareils utilisés pour multiplier le trafic transmis sont appelés amplificateurs ou réflecteurs [9]. Ces appareils sont généralement des hôtes légitimes qui ne savent pas qu'ils sont un réflecteur. Par rapport aux botnets, les réflecteurs ne sont généralement ni infectés ni contrôlés ; cependant, l'attaquant utilise les failles des protocoles et d'autres vulnérabilités afin de faire multiplier le trafic par ces appareils. Dans les attaques de réflexion, la relation requête-réponse est un principe clé [10]. Lorsque l'attaquant envoie une requête au réflecteur il lui répondra par un message réponse. L'attaquant envoie des paquets aux amplificateurs avec l'adresse IP usurpée de la victime. Les amplificateurs augmentent ensuite la quantité de paquets et / ou la taille des paquets et envoient la réponse à l'adresse IP usurpée. En raison d'une adresse IP falsifiée, tout ce trafic amplifié est dirigé vers la cible, ce qui pourrait entraîner un déni de service. Cela apporte plusieurs difficultés aux contre-mesures traditionnelles. Premièrement, l'origine de l'attaque est très difficile à retracer, en raison du fait que ce sont essentiellement les amplificateurs qui exécutent involontairement l'attaque. Par conséquent, la plupart des méthodes traditionnelles pourraient indiquer que les amplificateurs sont la source de l'attaque, ce qui n'est pas correct [4]. De plus, les attaquants disposant d'une faible bande passante pourraient effectivement multiplier leur trafic d'origine plusieurs fois. Les attaques d'amplification sont généralement utilisées dans deux scénarios possibles. Soit l'attaquant utilise son propre ordinateur pour multiplier son trafic, soit l'intrus utilise un botnet pour amplifier le trafic de toutes les machines botnet et le diriger vers la cible. Ce dernier pourrait conduire à des flux de trafic très importants.

2.3.4 Impacts des attaques DoS sur un réseau SDN

Due à la centralisation du contrôle dans l'architecture SDN, les attaques de DoS peuvent avoir des graves répercussions sur les performances du réseau conduisant à des cas où tout le réseau devient incapable à répondre aux besoins des utilisateurs. Ces attaques affectent la performance des trois éléments principaux dans le réseau SDN : le contrôleur SDN, la liaison entre le contrôleur et les commutateurs et le plan de transmission (les commutateurs et les liens du réseau).

Impact sur le plan de contrôle :

En cas d'une attaque de DoS, l'attaquant va envoyer une grande quantité de flux à travers le réseau SDN. Lorsque les commutateurs dans la couche infrastructure reçoivent ces nouveaux flux entrants, ils envoient des demandes au contrôleur pour obtenir des règles de commutation afin de les envoyer à la destination demandée. Par conséquent, le contrôleur SDN sera surchargé à cause de la quantité énorme de demandes provenant du plan de données du réseau, menant à des cas où le contrôleur devient totalement paralysé et ne puisse pas prendre aucune décision du routage.

Impact sur le plan de données :

Généralement, les commutateurs doivent enregistrer les règles de commutation dans leurs tables de commutation et les utiliser jusqu'à l'expiration des temporiseurs, l'idle timeout et le hard timeout. Dans une situation d'attaque de Dos, où l'attaquant inonde le

commutateur avec une quantité importante de flux, tout le trafic de données reçu par les commutateurs se traduit en règles de commutation fourni par le contrôleur, afin de les acheminer vers la destination. En effet, la mémoire TCAM du commutateur sera remplie par ces règles envoyées de contrôleur jusqu'à sa saturation. Lorsque cela se produit, les commutateurs sont forcés d'ajouter et de supprimer continuellement les règles de flux et d'envoyer plus des demandes vers le contrôleur ; cela engendre la congestion du plan de transmission ainsi qu'un retard dans le temps de transmission de données.

Impact sur la liaison contrôleur-commutateur :

Due à la communication agressive entre le contrôleur SDN et les commutateurs demandant des décisions de routage, la liaison entre le contrôleur et le commutateur (appelé aussi la bande passante du plan de contrôle) sera exténuée et congestionnée ; cela cause la perte de plusieurs messages «paquet-in» ainsi que le retard dans le temps de réponse des messages échangés entre le contrôleur et les commutateurs.

2.4 Système de Détection d’Intrusions (IDS)

2.4.1 Définition

Un IDS (Intrusion Detection System) est un outil ou un ensemble d’outils dont l’objectif est de surveiller le trafic entrant et sortant du réseau, dans le but de détecter une attaque ou une intrusion dans le système et déclencher différentes alertes en fonction de sa configuration. Un IDS analyse le réseau en temps réel, il nécessite donc des ressources matérielles.

Il est très facile de mettre en place une attaque du type DoS qui soit efficace. Pour se pré-munir de ces attaques, on doit pouvoir être capable de détecter de manière efficace une attaque. Cependant, il peut être difficile d’identifier un paquet licite d’un paquet provenant d’un attaquant. Mais il existe plusieurs outils qui permettent avec plus ou moins d’efficacité de détecter/bloquer une attaque.

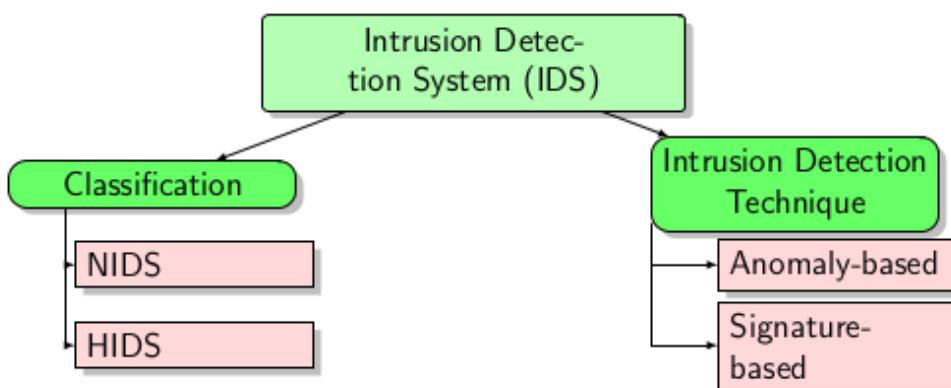


FIGURE 2.1 – Fonctionnement des systèmes de détection d’intrusions

2.4.2 Les approches

Il existe deux approches d'IDS :

- **Le NIDS** (Network Based IDS) assure la sécurité au niveau réseau. Il va donc écouter tout le trafic du réseau et générer des alertes en cas de comportement anormal.
- **Le HIDS** (Host Based IDS) réside sur une machine en particulier. Il surveille et analyse le trafic de la machine hôte pour déceler des intrusions ou des attaques (DoS par exemple).

Ces deux approches peuvent être fusionnées afin d'améliorer la précision de détection d'intrusion ; cette approche est appelée l'approche hybride.

2.4.3 Les méthodes de détection d'intrusions

Il existe trois méthodes principales de détection d'intrusions :

- **Détection par scénario** (Misuse Detection) : pour détecter les attaques, elle utilise une grande base de données qui contient les attaques connues et les associe aux caractéristiques des événements qui se produisent lors d'une attaque similaire. Cette technique est aussi appelée la détection basée sur la signature (Signature-Based Detection) ou la détection basée sur la connaissance (Knowledge-Based Detection).
- **Détection basée sur l'anomalie** (Anomaly-Based Detection) : utilise un profil qui représente les activités qui ne présentent aucune menace d'une éventuelle attaque ou une préparation d'attaque. Les événements sont par la suite comparés à ce profil d'activités.
- **Analyse de protocole avec état** (SPA) : commence par définir un ensemble de contraintes qui représente le comportement autorisé d'un programme ou un protocole donné. Le système surveille les opérations exécutées par le programme ou le protocole et vérifie s'ils respectent les contraintes définies par le système.

2.4.4 Les méthodes de collection d'informations sur les flux

Les IDS collectent les informations sur les flux en utilisant une de ces deux méthodes :

- **Première méthode** : Collecter les statistiques sur les flux calculés par les switches OpenFlow.
- **Deuxième méthode** : Capturer les paquets avec leur renfilage et extraire les informations contenues dans ces paquets.

A) Utilisation des statistiques OpenFlow :

La plupart des solutions de détection d'intrusions dans les SDN utilisent les statistiques calculées par les switches OpenFlow. Dans ce type de solution l'IDS est implémenté sous forme d'application SDN qui demande périodiquement les statistiques concernant les flux en utilisant le protocole OpenFlow. Les statistiques sont par la suite analysées au niveau de cette application SDN pour détecter l'intrusion. Les méthodes d'analyse sont basées sur des méthodes statistiques ou d'apprentissage automatique.

B) Utilisation de l'apprentissage automatique :

Le principe de ce type de solutions est d'utiliser un dataset (un ensemble de données) présentant des caractéristiques extraites de différents flux de paquets pour entraîner un système intelligent capable de prendre en entrée un vecteur de caractéristiques décrivant un flux de paquets et le classifier comme flux d'attaque ou flux bénins. L'efficacité de ce type de solutions est directement liée au choix du dataset, aux caractéristiques utilisées, et au modèle d'apprentissage automatique.

2.4.5 Évaluation des performances d'un NIDS

L'efficacité du NIDS est évaluée par plusieurs paramètres. Une matrice de confusion est un tableau qui est souvent utilisé pour calculer les performances d'un NIDS. Elle est décrite dans le tableau suivant.

Prédiction		
Classe réelle	Anomalie	Légitime
Anomalie	Vrai Positif (VP)	Faux Négatif (FN)
Légitime	Faux Positif (FP)	Vrai Négatif (VN)

TABLE 2.2 – Matrice de confusion

- VP : Le nombre d'enregistrements anomalies correctement classés.
- VN : Le nombre d'enregistrements normaux correctement classés.
- FP : Le nombre d'enregistrements normaux mal classés.
- FN : Le nombre d'enregistrements anomalies mal classés.

Afin d'évaluer le NIDS, les métriques suivantes sont calculées :

- **Exactitude (E)** : Indique le pourcentage de prédictions correctes sur la totalité des prédictions :

$$E = \frac{VP+VN}{VP+VN+FP+FN} \times 100\%$$

- **Précision (P)** : Indique combien d'intrusions prédites par un NIDS sont réellement des intrusions. Plus la précision est élevée, plus le taux de fausse alerte est faible :

$$P = \frac{VP}{VP+FP} \times 100\%$$

- **Rappel (R)** : Pourcentage des intrusions prédites par rapport à toutes les intrusions présentées :

$$R = \frac{VP}{VP+FN} \times 100\%$$

- **F1-Measure (R)** : Une mesure qui combine la précision et le rappel :

$$R = 2 \times \frac{1}{\frac{1}{P} + \frac{1}{R}} \times 100\%$$

- **Taux de faux Positif (TFP) et Taux de vrai Positif (TVP)** :

$$TFP = \frac{FP}{VN+FP} \times 100\% , TVP = \frac{VP}{VN+FP} \times 100\%$$

2.5 Travaux existants

Récemment, diverses solutions ont été proposées pour régler les problèmes de sécurité au sein des réseaux SDN. Les techniques d'apprentissage automatique sont largement appliquées pour améliorer l'efficacité de détection, y compris les réseaux neurone, les arbres de décision, les SVM (Support Vector Machine) et les réseaux bayésiens. L'intégration d'un système de détection d'intrusions dans l'architecture SDN reste la meilleure solution pour sécuriser l'environnement SDN. Comme mentionné dans la section 2.4.2, deux approches existent pour concevoir un IDS, une est basée machine (HIDS) et l'autre est basée réseau (NIDS), autrement appelé basé flux. Dans notre travail nous nous focaliserons sur les systèmes de détection basés flux, donc nous allons présenter en ce qui suit quelques IDS déjà existants afin de pouvoir faire une étude comparative ; technique d'apprentissage adoptée, les caractéristiques utilisées, le dataset, résultat de prédiction.

2.5.1 Flow based Intrusion Detection System [11] :

Les auteurs de ce travail ont proposé un système de détection basé flux pour détecter le trafic anormal circulant dans les réseaux SDN. Leur IDS est basé sur l'extraction d'un ensemble de caractéristiques prédéfinies de manière régulière chaque une seconde. Cette étape est suivie par l'agrégation des caractéristiques extraites pour pouvoir entraîner le modèle. La figure 2.2 illustre l'architecture de cette IDS.

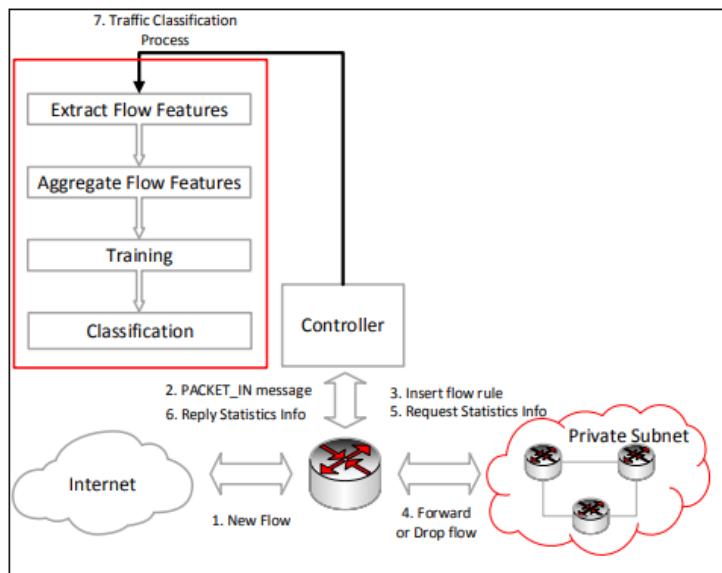


FIGURE 2.2 – IDS [11]

A) Extraction et agrégation des caractéristiques :

Les commutateurs OF envoient souvent des messages *Packet-In* au contrôleur quand aucune correspondance n'est trouvée dans la table de flux du commutateur. Chaque message contient les informations du premier paquet de chaque flux et quand il est reçu au niveau du contrôleur ces informations sont extraites et sauvegardées. En plus, ce qui est bien avec les SDN, ces commutateurs peuvent envoyer des statistiques pour chaque entrée de flux, ce qui permettra la collecte d'informations de flux chaque seconde.

B) Entraînement du modèle et la classification :

Les échantillons extraits de la phase d'agrégation sont utilisés dans un modèle de classification supervisé. Par conséquent, le classificateur formé attribuera des classes à chaque échantillon chaque seconde. Puisque chaque flux peut avoir plus d'une instance au cours de sa durée de vie, son comportement sera classé plusieurs fois. Le classifieur *Bagged-Trees* [12] a été utilisé pour construire le modèle.

2.5.2 Machine based Intrusion Detection System [13] :

Dans ce travail un modèle de réseaux de neurones a été intégré dans le système pour détecter les anomalies dans le trafic. Le système fonctionne comme suit; les commutateurs OF envoient périodiquement les statistiques de flux au contrôleur, ces statistiques vont être récupérées et traitées par le module responsable pour détecter le comportement d'anomalie dans le flux. Pattern recognition (la reconnaissance de motifs) est implémentée dans ce modèle pour classer les entrées dans un ensemble de catégories cibles. L'architecture de cet IDS se compose de trois couches : une couche d'entrée (Input Layer), une couche cachée (Hidden Layer) et une couche de sortie (Output Layer). L'algorithme "Backpropagation" est utilisé pour entraîner le modèle.

IDS	Machine based Intrusion Detection System [13]	Flow based Intrusion Detection System [11]
Technique d'apprentissage automatique	Neural Network	Bagged Trees
Caractéristiques utilisées	<ul style="list-style-type: none"> - Durée - Type de protocole - Nombre d'octets source-destination - Nombre d'octets destination-source - Nombre de connexions avec le même hôte - Nombre de connexions vers le même service 	<ul style="list-style-type: none"> - Durée de mesure - Nombre de paquets - Nombre d'octets - Variation dans la durée du flux - Variation dans le nombre de paquets - Variation dans le nombre d'octets
Dataset	NSL-KDD (standardisé) créé en 1999	Généré avec une émulation de l'environnement SDN (non standardisé) publiée en 2017
Type d'attaques dans le dataset	<ul style="list-style-type: none"> - DoS - Remote2Local - User2Root - Probe 	<ul style="list-style-type: none"> - DoS - Http brute force - SSH brute force

TABLE 2.3 – Comparaison entre les deux IDS proposés

Résultats expérimentaux

Taux de vrai Positif	0.941	0.9834
Taux de faux Positif	0.005	0.016
Exactitude	0.973	Non mentionnée

2.5.3 Comparaison entre ces deux IDS

Nous remarquons que les deux solutions donnent des résultats de performances très intéressants. Les auteurs de la solution [11] n'ont pas mentionné le taux de faux négatifs qu'est un élément très important dans l'évaluation d'un IDS. L'utilisation d'un dataset non standardisé dans [11] est discutable sur le plan de la validité des attaques, les flux générés et le bon déroulement de l'extraction des informations et caractéristiques lors de la création du dataset. La solution [13] utilise un ancien dataset(NSL-KDD créé en 1999) qui s'avère dépassé dû aux grands changements dans les réseaux, et l'apparition de nouveaux types d'attaques récentes. Par conséquent, l'IDS peut être inefficace face aux attaques récentes ainsi que dans la reconnaissance des caractéristiques des flux normaux modernes.

D'après cette étude nous constatons qu'il est nécessaire de mettre en place une solution de détection des attaques DoS dans les SDN qui prend en considération les critiques susmentionnées.

2.6 Conclusion

Dans ce chapitre nous avons tout d'abord étudié les problèmes de sécurité dans l'environnement SDN. Nous avons ensuite présenté quelques solutions déjà proposées pour détecter les attaques Dos. Dans le chapitre suivant, nous allons nous familiariser avec la technique d'apprentissage que nous utiliserons pour la conception de notre solution.

Chapitre 3

Data Mining

Pour construire n'importe quel modèle de prédiction, il est nécessaire de choisir une technique d'apprentissage automatique ainsi qu'un Dataset pour traîner le modèle. Dans ce chapitre nous allons parler sur les Dataset, comment choisir et évaluer un Dataset, nous parlerons aussi sur l'apprentissage automatique ainsi que ses deux grandes classes. Cette discipline de traitement de grandes quantité de données et choix de modèles et connue sous le nom de *Data-Mining*.

3.1 Data Minig

3.1.1 Définition

L'exploration de données, connue aussi sous l'expression de fouille de données, forage de données, prospection de données, a pour objet l'extraction d'un savoir ou d'une connaissance à partir de grandes quantités de données, par des méthodes automatiques ou semi-automatiques. Le processus le plus connu du Data-Mining est le KDD (knowledge discovery in database)[14], appelé en français extraction de connaissances à partir de données ou processus de découverte de connaissances.

3.1.2 Étapes du processus KDD

Le processus KDD est sur trois étapes comme suit :

- 1- **Préparation des données** : Elle comprend la collecte, l'intégration, la transformation, le nettoyage et la réduction des données. Dans notre cas la Collecte d'information est faite en capturant le trafic réseau à l'aide du *port-mirroring* des équipements réseau.
- 2- **Recherche des modèles** : Consiste à appliquer l'analyse des données et les algorithmes de découverte qui produisent une énumération de modèles sur les données.
- 3- **Évaluation des modèles** : Consiste à estimer l'erreur et la précision sur les modèles extraits. Un modèle est considéré comme une connaissance s'il dépasse un certain pourcentage de précision.

3.1.3 Dataset

Pour concevoir notre modèle d'apprentissage on a besoin d'une grande quantité de données qui représente plusieurs flux, malins et bénins, capturés dans un réseau. Le choix du bon Dataset contenant ce type de données est cruciale pour construire notre modèle de prédiction. Dans ce qui suit quelques propriétés à prendre en considération lors du choix d'un Dataset [15].

- **Year of Creation :** Étant donné que le trafic réseau est soumis à de nouveaux scénarios d'attaque apparaissent quotidiennement, l'âge d'un ensemble de données de détection d'intrusion joue un rôle important. Cette propriété décrit l'année dans laquelle le trafic a été capturé.
- **Normal User Behavior :** Cette propriété indique la disponibilité du comportement normal de l'utilisateur dans un ensemble de données et prend les valeurs **Oui** ou **Non**. La valeur **Oui** indique qu'il y a un comportement normal dans l'ensemble de données. En général, la qualité d'un IDS est principalement déterminée par son taux de détection des attaques et son taux de fausses alarmes. Par conséquent, la présence d'un comportement normal est indispensable pour évaluer un IDS.
- **Attack Traffic :** Un Dataset pour IDS devrait contenir divers scénarios d'attaque. Cette propriété indique la présence de trafic réseau malicieux dans le Dataset.
- **Kind of Traffic :** La propriété Type de trafic décrit trois origines possibles du trafic réseau : réel, émulé ou synthétique. Réel signifie que le trafic réseau réel a été capturé dans un environnement réseau en production. Émulé signifie que le trafic réseau réel a été capturé dans un banc d'essai ou un environnement réseau émulé. Synthétique signifie que le trafic réseau a été créé synthétiquement (par exemple, par un générateur de trafic) et non capturé par un dispositif réseau réel.
- **Balanced :** Pour la phase d'apprentissage les données doivent être équilibrées et respectent leurs étiquettes de classe. Cette propriété indique si le Dataset est équilibré. Un Dataset déséquilibré devrait être équilibrés selon des méthodes appropriées, avant de passer au algorithme d'apprentissage.
- **Labeled :** Un Dataset étiqueté est nécessaire pour traîner et tester les modèles d'apprentissage, supervisés et non-supervisés. Cette propriété indique si le Dataset est étiqueté ou non.

Dataset	Attaques
CICDoS [16]	Attaques DoS sur la couche d'application (ddosim, Goldeneye, hulk, RUDY, Slowhttptest,Slowloris).
CICIDS 2017 [17]	Botnet, DoS, DDoS, infiltration, SSH brute force, Injection SQL.
CIDDS-001 [18]	DoS, port scans, SSH brute force.
DDoS 2016 [19]	DDoS(HTTP Flood, ICMP Flood, UDP Flood).
CICIDS 2018 [20]	DoS, FTP and SSH brute force, Botnet, Infiltaration, DDoS.
CICDDoS 2019 [21]	Attaques DDoS Reflectives.

TABLE 3.1 – Quelques Datasets disponibles pour les système de détection d'intrusions

3.2 Apprentissage automatique

L'apprentissage automatique est un concept de développement, d'analyse et d'implémentation qui permet à une machine d'apprendre d'une manière automatique la résolution de problèmes qui s'avéraient complexe à traiter en utilisant les algorithmes classiques.

L'apprentissage automatique est utilisé pour résoudre plusieurs types de problèmes complexes dont la classification et la régression. Le problème de classification s'agit de classifier des données ou des instances en plusieurs classes selon les caractéristiques de ces derniers. Par exemple, apprendre à reconnaître si un flux est bénin ou malin est un problème de classification binaire ; décider le type d'une attaque parmi plusieurs types d'attaques possibles est un problème de classification multi-classes. Les problèmes de régressions sont des problèmes d'approximation des valeurs continues d'une certaine fonction. Par exemple, apprendre à prédire la température pour un jour donné connaissant les diverses conditions atmosphériques.

Les algorithmes d'apprentissage automatique sont classés sous deux grandes classes. Cette classification est directement liée à l'inclusion ou pas d'étiquettes dans l'ensemble de données.

3.2.1 Apprentissage supervisé

L'apprentissage supervisé est un système qui fournit à la fois les données en entrée et les données attendues en sortie. Les données en entrée et en sortie sont étiquetées en vue de leur classification, afin d'établir une base d'apprentissage pour le traitement ultérieur des données. Les systèmes d'apprentissage automatique supervisé alimentent les algorithmes d'apprentissage avec des quantités connues qui étayeront les futures décisions. Ils sont associés pour la plupart à une intelligence artificielle basée sur la récupération, mais ils peuvent aussi reposer sur un modèle d'apprentissage génératif.

Les données utilisées pour l'apprentissage supervisé sont une série d'exemples comprenant des paires composées de sujets en entrée et de sorties attendues. Les modèles d'apprentissage supervisé présentent certains avantages sur les modèles non supervisés, mais ils ont aussi des limites. Par exemple, ils sont plus susceptibles de prendre des décisions auxquelles les humains peuvent s'identifier parce qu'elles reposent sur des données fournies par les humains. Mais dans le cas d'une méthode basée sur la récupération, les systèmes d'apprentissage supervisé ont des difficultés à traiter les nouvelles informations. Si un système qui connaît deux catégories de flux, le flux d'une attaque SYN-Flooding et le flux bénin par exemple, reçoit un flux d'une attaque UDP-Flooding, il devra le placer dans l'une ou l'autre de ces deux catégories, ce qui sera incorrect. Alors que si le système était génératif, il ne saurait pas forcément reconnaître le type d'attaque UDP-Flooding, mais il serait capable de l'identifier comme appartenant à une autre catégorie.

Parmi les algorithmes d'apprentissage supervisé nous trouvons les machines à vecteur de support (SVM), AdaBoost (adaptative boosting), k plus proches voisins (k-NN) et les réseaux de neurones convergeant vers un état final précis.

3.2.2 Apprentissage non supervisé

L'apprentissage non supervisé consiste à apprendre à un algorithme d'intelligence artificielle (IA) des informations qui ne sont ni classées, ni étiquetées, et à permettre à cet algorithme de réagir à ces informations sans supervision.

Dans ce mode d'apprentissage, le système d'IA peut regrouper des informations non triées en fonction de leurs similitudes et de leurs différences, même si aucune catégorie n'est indiquée. Les systèmes d'IA capables d'utiliser l'apprentissage non supervisé sont souvent associés à des modèles d'apprentissage génératifs, mais ils peuvent aussi fonctionner avec une approche basée sur la récupération (souvent associée à l'apprentissage supervisé).

Les algorithmes d'apprentissage non supervisé peuvent exécuter des tâches de traitement plus complexes que les systèmes d'apprentissage supervisé, mais ils peuvent aussi être plus imprévisibles. Même si un système d'IA d'apprentissage non supervisé parvient tout seul, par exemple, à faire le tri entre des flux bénins et des flux malins, il peut aussi ajouter des catégories inattendues et non désirées pour y classer des flux inhabituels, créant la confusion au lieu de mettre de l'ordre. Parmi les algorithmes d'apprentissage non supervisé nous citons K-means clustering (K-moyenne), Dimensionality Reduction (Réduction de la dimensionnalité), Distribution models (Modèles de distribution) et Hierarchical clustering (Classification hiérarchique).

3.2.2.1 Clustering

Il s'agit essentiellement d'un type de méthode d'apprentissage non supervisé. Clustering est la tâche de diviser la population ou les points de données en un certain nombre de groupes de sorte que les points de données des mêmes groupes soient plus similaires aux autres points de données du même groupe et différents des points de données des autres groupes. Il s'agit essentiellement d'une collection d'objets sur la base de la similitude et de la dissemblance entre eux.

Le clustering est très important car il détermine le groupement intrinsèque des données non marquées. Il n'y a aucun critère pour un bon clustering. Cela dépend de l'utilisateur, quels sont les critères qu'il peut utiliser pour satisfaire ses besoins. Par exemple, nous pourrions être intéressés à trouver des représentants pour des groupes homogènes (réduction des données), à trouver des groupements utiles et appropriés ("useful" data classes) ou à trouver des objets de données inhabituels (outlier détection). L'algorithme du clustering doit faire quelques hypothèses qui constituent la similitude des points et chaque hypothèse fait des clusters différents et également valides.

3.2.2.2 Méthodes de clustering

Parmi les méthodes de Clustering on trouve :

- Méthodes basées sur la densité : ces méthodes considèrent les clusters comme la région dense ayant une certaine similitude et différence des autres régions denses de l'espace. Ces méthodes ont une bonne précision et la capacité de fusionner deux clusters. Exemples : DBSCAN, OPTICS ... etc.
- Méthodes basées sur la hiérarchie : les clusters formés dans cette méthode forment une structure de type arbre basé sur la hiérarchie. Les nouveaux clusters sont formés en utilisant ceux précédemment formés.
- Méthodes de partitionnement : ces méthodes partitionnent les objets en k clusters et chaque partition forme un cluster. Cette méthode est utilisée pour optimiser une fonction de similarité de critère objectif comme lorsque la distance est un paramètre majeur. Exemple : K-means, CLARANS ... etc.

3.2.2.3 K-Means

K-means est un algorithme non supervisé de clustering non hiérarchique. Il permet d'analyser un jeu de données caractérisées par un ensemble de descripteurs, afin de regrouper les données similaires en groupes (ou clusters). La similarité entre deux données peut être inférée grâce à la "distance" séparant leurs descripteurs ; ainsi deux données très similaires sont deux données dont les descripteurs sont très proches. Cette définition permet de formuler le problème de partitionnement des données comme la recherche de K "données prototypes", autour desquelles peuvent être regroupées les autres données. Ces données prototypes sont appelées centroïdes ; en pratique l'algorithme associe chaque donnée à son centroïde le plus proche, afin de créer des clusters. D'autre part,

les moyennes des descripteurs des données d'un cluster, définissent la position de leur centroïde dans l'espace des descripteurs : ceci est à l'origine du nom de cet algorithme (K-means ou K-moyennes en Français). Après avoir initialisé ses centroïdes en prenant des données au hasard dans le jeu de données, K-means alterne plusieurs fois ces deux étapes pour optimiser les centroïdes et leurs groupes :

- 1- Regrouper chaque objet autour du centroïde le plus proche.
- 2- Replacer chaque centroïde selon la moyenne des descripteurs de son groupe.

Après quelques itérations, l'algorithme trouve un découpage stable du jeu de données : on dit que l'algorithme a convergé. Comme tout algorithme, K-means présente des avantages et des inconvénients : il est simple, rapide et facile à comprendre ; cependant il ne permet pas de trouver des groupes ayant des formes complexes.

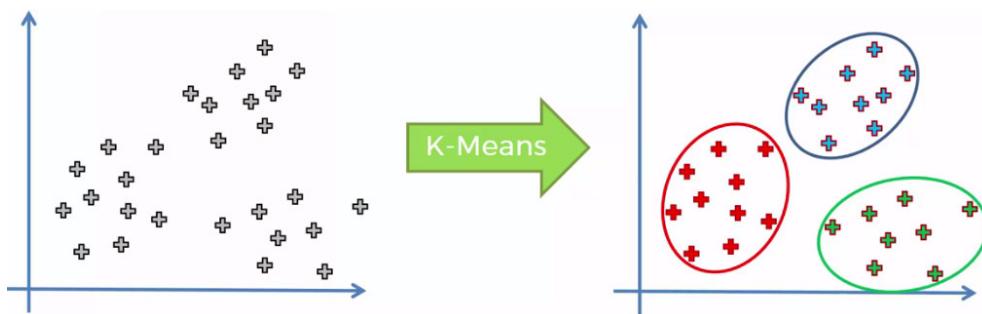


FIGURE 3.1 – K-Means

3.2.3 Conclusion

Dans ce chapitre nous avons discuté sur le Data-Mining, en commençant par définir c'est quoi cette discipline, pour passer par la suite au dataset en exposant les majeures propriétés à considérer lors du choix d'un dataset. L'apprentissage automatique était la dernière section de ce chapitre, où on a présenté les différentes méthodes et algorithmes d'apprentissage existants.

Chapitre 4

Conception

le Software-Defined Networking a rapidement émergé comme une technologie prometteuse pour les réseaux futurs et a gagné beaucoup d'attention. Cependant, la nature centralisée du SDN rend le système vulnérable aux attaques par déni de service (DoS), une fois le contrôleur compris, tout le réseau cessera de fonctionner. Mais cette centralisation a un avantage, la gestion centralisée des équipements réseau, elle permet d'avoir une vue globale des flux de trafic, ce qui offre un meilleur système de défense contre les attaques DoS.

Comme mentionné dans la section 2.3.3, nous nous intéressons dans notre travail à une attaque DOS spécifique, connue sous le nom de **Reflective-DoS** (RDos). Cette attaque est un peu spéciale et diffère carrément des autres types d'attaques Dos, dans le principe de fonctionnement et les dommages causés. On parlera plus sur cette attaque dans la section suivante.

Ce chapitre portera sur la conception de notre solution. Nous commençons par la problématique et la motivation de notre travail, suivi des hypothèses de conception. On passera après à la première étape de la conception, où on présentera l'architecture de notre système et ses différents composants. L'étape suivante est la construction de notre modèle de clustering et terminera évidemment avec une conclusion sur le travail effectué.

4.1 Problématique

Mars 2018, un nouveau record a été marqué avec 1.7 Tbps de trafic généré par une attaque DoS réfléctrice. La compagnie **Arbor Networks** a affirmé que son système d'analyse de trafic, ATLAS, a enregistré 1.7 Tbps d'une attaque reflective contre un site web d'un client[22].

RDos n'attaque pas directement la cible mais envoie plutôt plusieurs requêtes vers un service tiers exploitable (c. -à-d. le réflecteur, généralement c'est un serveur) avec une adresse IP d'expéditeur usurpée, ce qui rend l'attaquant anonymat. Les réponses du serveur tiers sont ensuite envoyées à la cible d'attaque réelle et causer une surcharge. Les protocoles avec des messages de réponse qui sont beaucoup plus grands que les messages de demande sont particulièrement bien adapté à ces attaques en raison des effets d'amplification. La nature de ces attaques nécessite des services qui fonctionnent sans connexion établie entre le client et le serveur. Une étude récente[23] a trouvé que

99.72% des attaques RDoS utilisent des protocoles basé UDP, comme DNS, NTP, ..etc. Les messages reçus dans une attaque RDoS sont difficiles à différencier du trafic bénin, car ils sont conformes aux spécifications du protocole. Les réflecteurs sont correctement gèrent toutes les demandes comme légitimes, mais l'absence de réponse est une caractéristique des attaques réflectives qui ne peut être masquée.

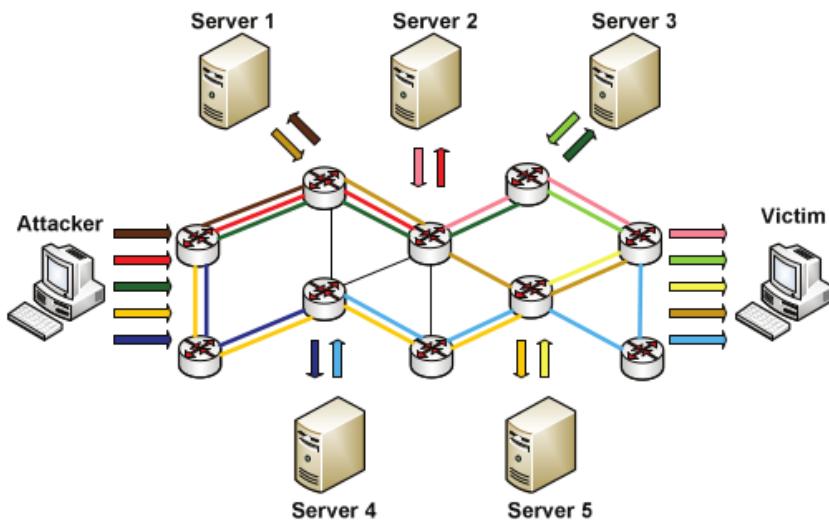


FIGURE 4.1 – Principe de dénie de service réfléctif

À ce niveau, on peut clairement voir que l'impact de ce type d'attaque est en double ; en premier lieu, la consommation de la bande passante des liens réseau à cause du nombre excessif des messages requêtes et réponses échangés. En deuxième lieu, la surcharge des serveurs tiers avec des messages requêtes, qui vont les traiter évidemment, car pour eux ils paraissent des requêtes légitimes.

4.2 Motivation du travail

Un article[24] sur les travaux connexes a montré que la défense contre les attaques RDoS peut être divisée en 3 principales tâches : (1) surveillance (*monitoring*) et la collecte de données du trafic réseau, (2) l'interprétation de ces données et la détection d'une attaque en cours, et (3) la mitigation de l'attaque. Selon leurs tâches principales, les solutions existantes varient considérablement dans leurs hypothèses et leurs exigences. Dans la littérature on trouve que la surveillance du trafic, qui représente la première phase du processus de détection, se fait sur ces trois couches :

- La couche réseau.
- La couche de transport.
- La couche d'application.

Par exemple *PacketScore*, par Kim et al [25], utilise les statistiques de paquets collectés pour comparer chaque paquet reçu au trafic bénin et d'attaque pour ensuite lui attribuer une note. Les paquets qui sont similaires au trafic d'attaque sont éliminés. Cette approche nécessite une surveillance active de la couche réseau. Plusieurs autres travaux [26, 27] s'intéressent à la couche de transport pour la détection des anomalies en analysant les paquets de transport, ICMP, TCP et UDP.

Pour cette fin, nous proposons une solution, basée sur une approche de clustering, pour la détection des attaques Dos réflectives dans un réseau SDN. Cette solution prend en charge aussi les attaques DoS du type UDP-Flooding (voir section ??). La mitigation de l'attaque ne fait pas par partie de notre travail, on fera juste la détection. La première tâche, surveillance et collecte de données, va être lancée sur la couche de transport, où on surveillera spécialement le protocole de transport UDP, vu qu'il est le plus utilisé dans les attaques RDoS. Pour la deuxième phase, qui est la détection, on utilisera notre modèle d'apprentissage pour analyser les données collectées dans la phase précédente et décider si c'est une attaque ou non.

4.3 Présentation de la Solution

Les caractéristiques inhérentes des attaques de RDoS représentent un défi pour leur détection. Une surveillance et analyse continues du trafic réseau sont nécessaires pour la détection des flux malins. La solution qu'on propose, pour ce fait, est un système de détection, appelé **F-DoS**, qui sera déployée dans le réseau cible pour assurer la surveillance du trafic au but de détecter les attaques DoS (RDoS et UDP-Flooding).

Ces deux fonctions de surveillance et de détection sont assuré par deux modules intégrés dans notre système. Le premier est le module d'extraction d'informations, qu'on référencera par l'acronyme **MEI**. Une copie de chaque flux circulant dans le réseau est envoyée à ce module afin d'extraire ses caractéristiques. Les caractéristiques extraites sont ensuite envoyées au deuxième module de notre système, appelé **F-Clustering**, qui est un modèle intelligent, sa fonction est de grouper les flux en clusters, cluster des flux normaux et cluster des flux malins.

L'architecture et le principe de fonctionnement de notre système sont illustrés dans la figure suivante :

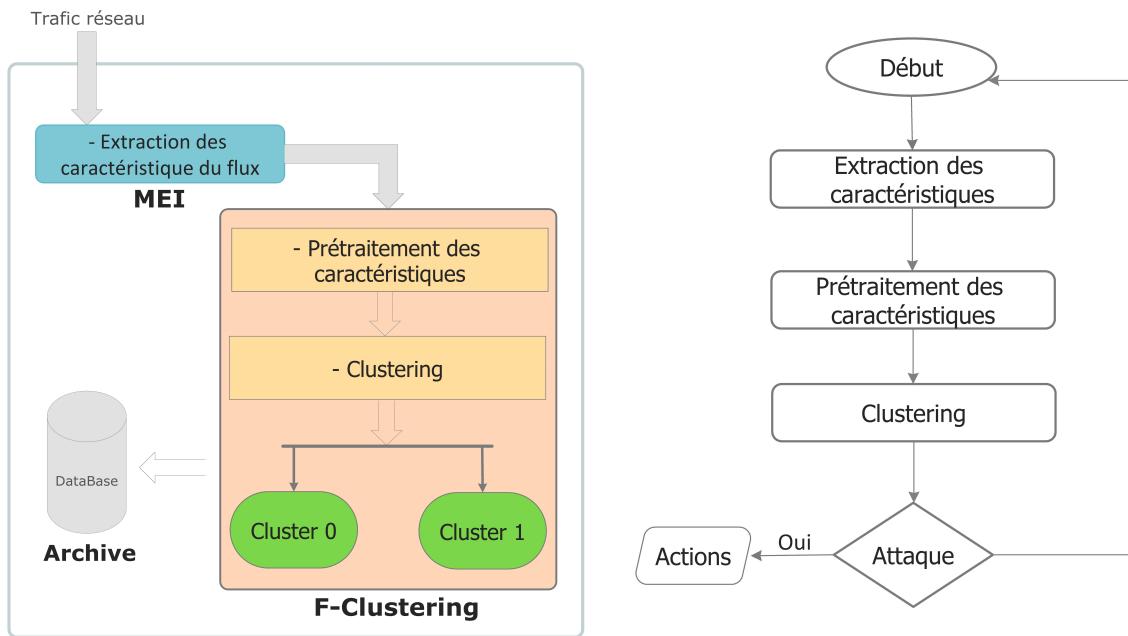


FIGURE 4.2 – Architecture et fonctionnement de F-DoS

4.3.1 Module MEI

Le rôle principal de ce module est l'extraction des informations des flux de données. Mais pour ce faire, ce module doit assurer trois fonctions : l'écoute passive, l'analyse et l'extraction. Ce module à un port dédié à l'écoute, qui est toujours active pour recevoir une copie de chaque paquet à l'aide de la technique de mise en miroir *Port Mirroring*. L'ensemble des paquets capturés est envoyé par la suite à l'analyseur. L'analyse des paquets n'est pas une fonction simple, elle doit être effectuée soigneusement, si on analyse mal les paquets on risque de fausser le résultat de détection. Cette fonction est compliquée a raison que, l'analyseur reçoit tous les paquets qui circulent dans le réseau, et on a dit précédemment qu'on s'intéresse aux informations des flux de données, un flux est un ensemble de paquets qui partage le même *Packet Header*(c-a-d même adresse source, adresse destination, port, ... etc). Donc l'analyseur doit calculer les flux en regroupant les paquets communs, identifier les flux de données susceptibles et éliminer tout autre flux non intéressant, comme le flux des messages de contrôle, le flux des messages de sollicitation,...etc. Les flux de données formés vont ainsi être passés à la dernière fonction de ce module pour extraction de caractéristiques de chaque flux.

4.3.2 Module F-Clustering

L'objet de notre travail est de proposer une approche de Clustering pour la détection des attaques Dos. On s'est penché vers le K-Means, nous l'avons amélioré et adapté à notre utilisation pour construire notre modèle intelligent qui est le coeur de ce module. Ce module prend en entrée le vecteur de caractéristiques construit par le module MEI, et d'après ses préconnaissances, et attribue le flux reçu à un des clusters préexistants. Si le flux est groupé dans le cluster des flux malin on dit qu'une attaque DoS (RDoS ou UDP-Flooding), on peut ainsi lancer une alerte et c'est à l'administrateur réseau de prendre action.

4.4 Conception du module MEI

Pour concevoir ce module, nous étions dans l'obligation de choisir un outil d'analyse et de calcul des caractéristiques des flux. Parmi les différents analyseurs existants, nous avons choisi ARGUS[31] pour plusieurs raisons :

- Son architecture client/serveur, est simple et personnalisable.
- Assure les trois fonctions de notre module MEI.
- Fonctionne en temps réel ce qui permet, par la suite, une détection d'attaques en temps réel.

Pour mettre en oeuvre ce module nous avons procédé comme suit :

- 1- Premièrement, une copie du trafic réseau est redirigée vers ce module à l'aide du *Port-Mirroring* activé au niveau des *switches*.
- 2- Un filtrage est appliqué pour supprimer tout paquet non intéressant pour ce travail.
- 3- Nous avons lancé le serveur ARGUS pour capturer les flux.
- 4- Nous avons lancé l'outil ARGUS-client, qui nous permet de sélectionner les caractéristiques et les informations des flux à extraire.
- 5- Nous traitons par la suite les informations collectées et nous les envoyons au module **F-Clustering**.

L'architecture interne du module MEI est illustrée dans la figure 4.3

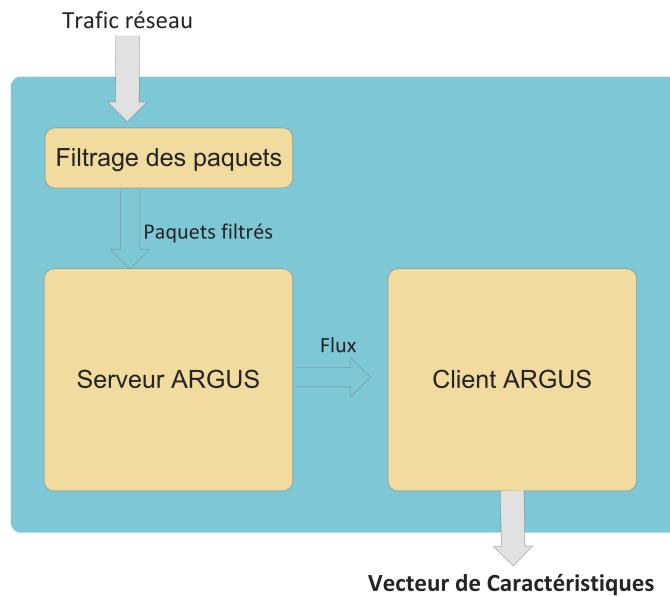


FIGURE 4.3 – Architecture du module MEI

4.5 Conception du module F-Clustering

Pour concevoir ce module nous avons suivi les étapes du processus KDD décrites dans la section 3.1.2 :

- Préparation des données : premières étapes du processus, où on parlera de notre source donnée et les différentes opérations effectuées sur ces données pour les préparer à la deuxième phase du processus.
- Recherche du modèle : le modèle adopté est un modèle d'apprentissage non supervisé, utilisant une approche de Clustering, qui va être trainé sur les données collectées dans la phase précédente.
- Évaluation du modèle : on fera l'évaluation du modèle dans le chapitre 5.

4.5.1 Préparation des données

A) Choix du dataset

Comme source de données, nous avons exploité le dataset **CICDDoS 2019[21]**. CICD-DoS2019 contient des flux bénins et des attaques Ddos les plus courantes, qui ressemblent aux vraies données du monde réel. Ce dataset contient différentes attaques DoS modernes réflectives telles que Portmap, Netbios, LDAP, UDP, TFTP, NTP. La période de capture a commencé le 12 janvier à 10 h 30 et s'est terminée à 17 h 15. Des attaques ont par la suite été exécutées au cours de cette période.

CICDDoS2019 est formé de plusieurs fichiers, chacun propre à une attaque. Vu la taille énorme de ces fichiers, et pour raison de simplifier le travail et pouvoir simuler l'architecture générale de notre réseau, on s'est mis d'accord sur le choix d'un seul fichier qu'on travaillera sur. Le fichier qu'on a choisi est propre à l'attaque TFTP, qui est lui-même un dataset contenant *** lignes et 87 colonnes (aussi référencées par caractéristiques ou attributs).

B) Prétraitement des données

Le dataset TFTP n'est pas propre, il contient des données avec des ordres de grandeur différents, des valeurs erronées, des dupliques, il n'est pas balancé et contient des attributs qu'on n'a pas besoin dans notre travail. Un prétraitement est nécessaire pour préparer le dataset en vue de l'utiliser dans l'étape de création du modèle d'apprentissage. Ce processus de prétraitement est sur trois étapes ; la première est le nettoyage du dataset, connue dans la littérature sous le nom de **Data Laundry**. Deuxième étape, sélection des attributs. Troisième étape, mise à l'échelle et balancement du dataset.

B-1) Data Laundry

Cette étape consiste à nettoyer le dataset :

- Les lignes contenant des valeurs "*Infinit*" ou "*Nan*" seront supprimées.
- Suppression des dupliques.
- Les données vont être formatées en un *datatype* standard.

B-2) Sélection des attributs

La qualité des données de l'ensemble d'apprentissages et des attributs qui les caractérisent forment le succès de la technique de clustering utilisée. La présence de variables redondantes, bruitées et peu corrélées avec le cluster d'un exemple rend le processus d'apprentissage plus difficile.

La détection d'intrusion est le genre d'application pour laquelle la sélection d'attributs peut être bénéfique car elle permet tout au moins d'aboutir aux mêmes performances de reconnaissance à partir d'un nombre de caractéristique moindres.

Nous avons choisi les attributs en analysant les propriétés intrinsèques des données, nous avons éliminé la plupart des attributs de base telles que les hôtes source et destination, les ports, les flags, ...etc. Nous avons éliminé aussi les attributs de contenu qui sont construits à partir de la charge utile (data) des paquets de trafic tels que le nombre d'échec de connexion. Car, ce sont des valeurs statiques et n'ont aucun impact sur la détection des attaques.

Le choix final des attributs était : le protocole pour filtrer le trafic et étudier le cas du protocole UDP uniquement, la durée du flux, la taille et le nombre de paquets envoyées et reçus, ainsi que les débits de transfert. Le tableau suivant décrit chacun de ces attributs.

Attribut	Déscription
Protocol	Protocol de transport
Flow Duration	Durée du flux
Total Fwd Packets	Nombre de paquets envoyées
Total Backward Packets	Nombre de paquets reçus
Total Length of Fwd Packets	Taille totale des paquets envoyées
Total Length of Bwd Packets	Taille totale des paquets reçus
Flow Bytes/s	Nombre de Byte par seconde
Flow Packets/s	Nombre de paquets par seconde
Fwd Packets/s	Nombre de paquets envoyées par seconde
Bwd Packets/s	Nombre de paquets reçus par seconde

TABLE 4.1 – Description des attributs

B-3) Mise à l'échelle et balancement

La mise à l'échelle consiste à diminuer la grande différence entre les valeurs d'une même caractéristique C. Nous avons utilisé la technique Min-Max Scaler sur ce dataset, qui pour chaque valeur X applique l'opération suivante :

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Où :

X : valeur à mettre à l'échelle de la caractéristique C.

X_{min} : plus petite valeur observée de la caractéristique C.

X_{max} : plus grande valeur observée de la caractéristique C.

Le dataset TFTP n'est pas balancé, il contient beaucoup plus de flux attaqué que de flux bénin. Si on l'utilise sans le balancer, notre modèle va toujours pencher vers le cluster des attaques. L'idée du balancement est de générer de nouveaux échantillons de la classe minoritaire à l'aide de fonctions mathématiques existantes. On parlera sur la bibliothèque python utilisée pour le balancement du dataset dans le chapitre ??.

4.5.2 Recherche du modèle

Deuxième étape du processus KDD. On a opté pour un modèle de Clustering, où a fait l'implémentation de l'algorithme K-Means avec des modifications pour l'adapter à notre utilisation. Le modèle prend en entrée le dataset construit dans l'étape de prétraitement, applique l'algorithme de clustering pour construire les deux clusters, attaque et bénin. En sortie, on aura un objet, qui groupe un flux donné dans un des clusters existants. Cet objet n'est rien d'autre que le module **F-Clustering** de notre système.

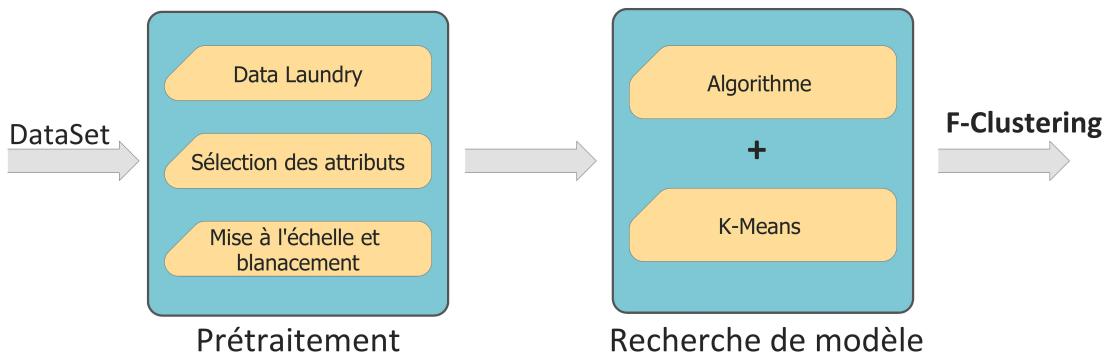


FIGURE 4.4 – Étapes de conception de F-Clustering

4.6 Module d'archivage

Nous avons pensé à rajouter, comme fonctionnalité additionnelle à notre système, un module d'archivages qui sauvegarde des logs datés et classés par ordres chronologiques sur les caractéristiques et les informations collectées sur les flux de données. Une étiquette, qui porte la valeur "ATTACK" ou "BENIGN", sera ajoutée à chaque flux, par le modèle de Clustering, pour l'identifier. 57 caractéristiques seront calculées et sauvegardées dans ce module, on trouve : la durée du flux, les adresses source et destination, Timestamp (heure de capture), protocole, ports source et destination, et bien d'autres. Ce module peut servir à plusieurs besoins, par exemple l'analyse périodique du système, il peut servir comme une source de données (dataset) pour d'autres projets qui nécessitent des données fiables et capturées dans un réseau réel.

4.7 Conclusion

Ce chapitre a porté entièrement sur la conception de notre solution, nous avons commencé par définir la problématique des attaques de déni de service réflectives et décrire la motivation de notre travail. Notre contribution dans ce domaine était un système de détection d'attaques DoS, nous avons vu son architecture générale et détaillé le rôle de chacun des modules qui le composent. Nous avons décrit son fonctionnement et présenté les différentes étapes de conception.

Dans le prochain chapitre, nous allons entamer la partie d'implémentation de notre système et son déploiement dans un réseau SDN simulé. Des tests expérimentaux seront aussi faits pour des raisons d'évaluation du système.

Chapitre 5

Implémentation, simulation et résultats

Au cours de ce chapitre nous allons présenter l'implémentation et le test de notre système dans un réseau SDN simulé. On commence par voir l'environnement de travail, les outils de simulation utilisés, pour passer ainsi à la phase des tests, qui est divisée en deux : premièrement nous allons tester notre modèle de clustering sur un jeu de données pour calculer les différentes métriques de performances citées dans la section 2.4.5. L'autre partie des tests concerne notre système de détection, où nous allons définir plusieurs scénarios d'attaque et de flux normaux, pour voir son comportement et est-ce qu'il est capable d'attribuer chaque flux au bon cluster.

5.1 Environnement de travail

Cette partie décrit l'environnement dans lequel notre solution sera déployée en présentant le contrôleur SDN choisi et l'outil de simulation du réseau utilisé.

5.1.1 Contrôleur SDN

Parmi les différents contrôleurs SDN qui existent nous avons opté pour travailler avec **Ryu**[4] vu la simplicité d'utilisation qui offre, en plus il est basé sur *Python* et supporte la majorité des versions d'OpenFlow.

Ryu est un contrôleur de réseau défini par logiciel (SDN) conçu pour augmenter l'agilité du réseau en le rendant facile à gérer et adapte la façon dont le trafic est traité. Il fournit des composants logiciels, avec des interfaces de programme d'applications (API) bien définies, qui rendent facile pour les développeurs de créer de nouvelles applications de gestion et de contrôle du réseau. Cette approche par composants aide les organisations à personnaliser leur déploiement pour répondre à leurs besoins spécifiques ; les développeurs peuvent rapidement et facilement modifier les composants existants ou mettre en œuvre leurs propres pour s'assurer que le réseau sous-jacent peut répondre aux demandes changeantes de leurs applications.

Pour télécharger et installer Ryu sur une machine UBUNTU :

```
% git clone git://github.com/faucetsdn/ryu.git
% cd ryu
% pip install .
```

5.1.2 Simulation du réseau SDN

Afin de pouvoir déployer notre système, on est obligé de simuler un réseau SDN réel avec tous ces équipements, contrôleurs SDN, commutateurs, hôtes, liens physiques. Ceci est possible à l'aide d'un émulateur réseau. Il en existe beaucoup d'émulateurs réseau, celui qui nous convient dans notre travail est *Mininet*[28].

Mininet

Mininet est un émulateur de réseau qui crée un réseau d'hôtes virtuel, de commutateurs, de contrôleurs et de liens. Les hôtes Mininet exécutent le système *Linux* standard, ce qui donne la possibilité d'exécuter des programmes *python* au niveau des hôtes. Les commutateurs Mininet prennent en charge Openflow pour un routage personnalisé hautement flexible.

Mininet est vaguement recommandé car :

- Est rapide, démarrer un réseau simple ne prend que quelques secondes.
- Vous pouvez créer des topologies personnalisées : un commutateur unique, de plus grandes topologies du type Internet, un centre de données, ou toute autre chose.
- Vous pouvez personnaliser le transfert de paquets : les commutateurs de Mininet sont programmables à l'aide du protocole Openflow.
- Comprend une interface de ligne de commande **CLI** pour le débogage ou l'exécution de tests sur réseau.

Pour installer mininet sur une machine UBUNTU :

```
% sudo apt install mininet
```

Ou bien :

```
% git clone git://github.com/mininet/mininet  
% mininet/util/install.sh
```

Simulation d'un simple réseau SDN

Ci-dessous un exemple sur comment lancer la simulation d'un simple réseau SDN avec un contrôleur, un switch OpenFlow et deux hôtes.

1- Lancer le contrôleur Ryu :

```
% ryu-manager ryu_controller.py
```

2- Créer la topologie avec mininet :

```
% sudo mn --controller remote --topo single,2 --switch ovs --mac
```

Terminal - topologie créée par mininet

```
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

5.2 Langage et Librairies utilisées

5.2.1 Python

Python est un langage de programmation très puissant et adaptable à tout type d'utilisation grâce à ces bibliothèques spécialisées, utilisé particulièrement comme un langage de script. Il est trop utilisé dans la programmation réseau et spécialement dans les réseaux SDN.

5.2.2 Scikit-learn

Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique. Elle est développée par de nombreux contributeurs² notamment dans le monde académique par des instituts français d'enseignement supérieur et de recherche comme Inria³. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support. Elle est conçue pour s'harmoniser avec d'autres bibliothèques libres Python, notamment NumPy et SciPy.[29]

5.2.3 Pandas

Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles. Les principales structures de données sont les séries (pour stocker des données selon une dimension - grandeur en fonction d'un index), les DataFrames (pour

stocker des données selon 2 dimensions - lignes et colonnes), les Panels (pour représenter des données selon 3 dimensions, les Panels4D ou les DataFrames avec des index hiérarchiques aussi nommés MultiIndex (pour représenter des données selon plus de 3 dimensions). [30]

5.2.4 Argus

Argus[31] est le premier système de flux réseau, développé par Carter Bullard au début des années 1980 à Georgia Tech. La technologie de flux réseau est devenue un élément essentiel de la cybersécurité moderne et Argus est utilisé dans certains des réseaux les plus importants du monde. Le système Argus tente de résoudre un grand nombre de problèmes liés aux données de flux réseau : échelle, performance, applicabilité, confidentialité et utilité.

5.3 Réalisation du module F-Clustering

On rappelle que ce module est composé d'un modèle intelligent qui utilise une approche de clustering pour la détection des attaques. Dans la section 4.5, nous avons décrit les étapes à suivre pour construire notre modèle de clustering qui sont accomplies à l'aide des deux scripts python, "Preprocessor.py" et "Cluster.Py".

Algorithm 1 Preprocessor.py

```

1- def cleanDataFrame(df) :
2-     df.drop('Infinite', 'NaN')
3-     df.drop_duplicates()
4-     df.format_types()
5-     return df

6- def selectAttributs(df):
7-     return df[['Protocol', 'Flow Duration', 'Total Fwd Packets',
       'Total Backward Packets', 'Total Length of Fwd Packets',
       'Total Length of Bwd Packets', 'Flow Bytes/s', 'Flow Packets/s',
       'Fwd Packets/s', 'Bwd Packets/s', 'Label']] 

    # preprocess the data frame
8- def preprocessor (df):
9-     df = cleanDataFrame(df) #Data laundry
10-    df = selectAttributs(df) #select necessary attributs
11-    df = balanceDataFrame(df)
12-    df = scaleDataFrame(df)
13-    return df

#end

```

Algorithm 2 Cluster.py

```

1- from Preprocessor.py import preprocessor
2- from sklearn.cluster import KMeans

4- df = read_csv('TFTP.csv')

5- newDF = preprocessor(df)

6- centers = calculateCenters(newDF) #returns 2 elements array

7- F_Clustering = KMeans(n_clusters=2, init=centers).fit(newDF)

```

5.4 Évaluation et test des performances

Dans cette partie, nous allons entamer la dernière étape du processus KDD qui est l'évaluation du modèle. Comme jeu de test nous avons sélectionné le dataset « TFTP » qui contient 1.315.348 flux étiquetés (attaque et bénini). Nous avons comparé les différents résultats obtenus par notre modèle de clustering avec les données initiales du dataset pour calculer les mesures de performances du modèle.

Les résultats des tests de performance étaient les suivants :

	Attaques	Bénins	Total
Attaques	657674	0	657674
Bénins	8887	648787	657674

TABLE 5.1 – Matrice de confusion

Exactitude	Précision	Rappel	F1-Measure	Taux de fausses alertes
99.3%	98.6%	100%	99.3%	1.3%

TABLE 5.2 – Mesures de performance

Nous remarquons que les résultats sont très satisfaisants. L'élément-clé qui nous a permis d'avoir tels résultats est le prétraitement du dataset, plus spécialement l'opération de normalisation et l'opération balancement. Pour montrer leur impact sur les performances du modèle nous allons faire une petite expérience, où nous allons annuler une opération de prétraitement à la fois, soit la normalisation, soit le balancement et calculer ensuite les mesures de performance du modèle.

Les résultats obtenus sont dans le tableau 5.3 et la figure 5.1 ;

Opération(s) effectuée(s)	Exactitude	Rappel	F1-Measure
Aucune	25.4%	24.4%	39.2%
Normalisation seulement	29.6%	27.6%	43.2%
Balancement seulement	71.5%	43.8%	60.6%
Balancement + Normalisation	99.3%	100%	99.3%

TABLE 5.3 – Impact des prétraitements sur les performances

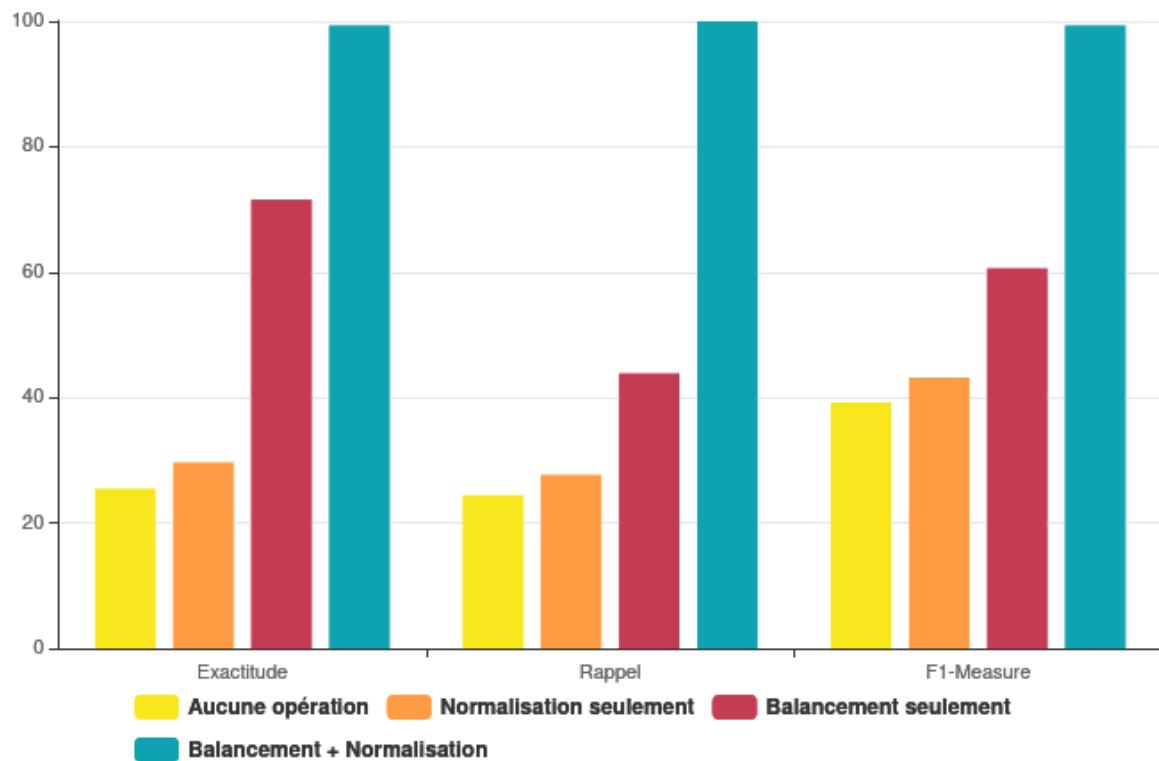


FIGURE 5.1 – Comparaison entre les résultats obtenus selon l'opération de prétraitement effectuée

En analysant l'histogramme ci-dessus on voit clairement la différence si on applique juste une seule opération de prétraitement à la fois, ou bien les deux opérations combinées. On choisissant la bonne méthode de normalisation ainsi que celle de balancement, nous avons pu faire passer le taux d'exactitude de notre modèle de **25.4%** à **99.3%**. Le choix des méthodes était selon une stratégie étudiée et une synthèse des résultats de plusieurs tests effectués sur notre dataset. Les bonnes performances d'un modèle sont toujours engendrées par un bon prétraitement des données.

5.5 Simulation

La simulation sera faite à l'aide de Mininet(2.2) qui est installé à coté de Ryu(4.3) sur une machine UBUNTU 20.04 LTS dotée d'un processus i5-6200U et 8 Go de RAM. Le réseau qu'on veut simuler contient les éléments suivants : un contrôleur SDN, un switch Open Flow, un serveur de fichiers, un serveur d'applications, 3 hôtes (2 hôtes attaquants at un hôte victime), et essentiellement notre système détection F-DoS.

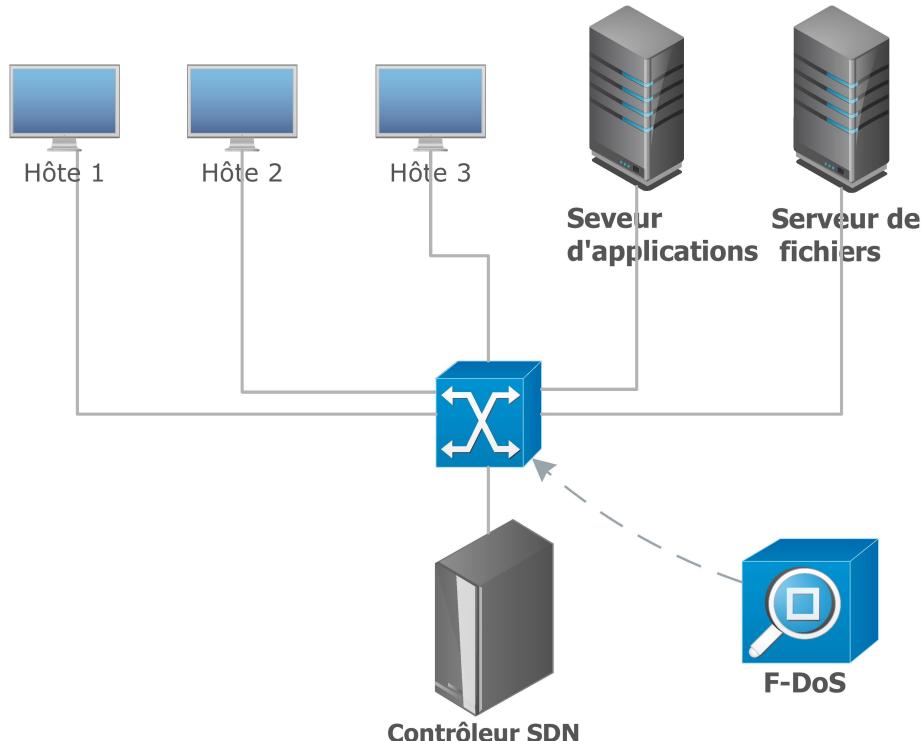


FIGURE 5.2 – Architecture de notre réseau SDN

5.5.1 Création de la topologie

Sur mininet nous allons créer la topologie de notre réseau constitué d'un contrôleur SDN, un switch OpenFlow, et 6 hôtes qui jouent les rôles illustrés dans le tableau 5.4;

Commande de création :

```
% sudo mn --controller remote --topo single,6 --switch ovs --mac
```

Mais faut d'abord lancé le contrôleur Ryu avec :

```
% ryu-manager ryu_controller.py
```

Hôte	Rôle(s)	@IP
h6	Système de détection F-DoS	10.0.0.6
h5	Serveur d'applications	10.0.0.5
h4	Serveur de fichiers	10.0.0.4
h3	Hôte victime	10.0.0.3
h2	Client de h5 ou attaquant	10.0.0.2
h1	Client de h4 ou attaquant	10.0.0.1

TABLE 5.4 – Rôle de chaque hôte dans la topologie mininet

```

Terminal - reda@reda-Latitude-5490:~ 
File Edit View Terminal Tabs Help
reda@reda-Latitude-5490:~$ sudo mn --controller remote --topo single,6 --switch ovs --mac
[sudo] password for reda:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 
```

FIGURE 5.3 – Création de la topologie sous mininet

La figure 5.5 indique que la topologie a été créés avec succès et la console mininet est active. On peut voir le contrôleur **c0**, le switch **s1** et les différents liens établis (h1, s1), (h2, s1), etc. L'étape suivante consiste à configurer chaque hôte de notre topologie.

5.5.2 Installation des fonctions

Jusqu'à présent les hôtes créés par mininet, ainsi que les switchs OF, n'ont que les fonctions de base définies par mininet. Donc on doit passer par tout hôte, à l'exception de **h3**, pour installer les fonctions désignées dans le tableau ???. Sur le switch, nous allons juste activer la fonction *Port-Mirroring* comme suit :

```

root@reda-Latitude-5490:~# ovs-vsctl -- --id=@m create mirror name=mymirror --
add bridge s1 mirrors @m
d8ebb36b-5e8e-416b-aab5-02855ae757d9
root@reda-Latitude-5490:~# ovs-vsctl -- --id=@"s1-eth6" get port "s1-eth6" -- s
et mirror mymirror select_all=true output-port=@"s1-eth6"
root@reda-Latitude-5490:~# ovs-vsctl list mirror mymirror
_uuid          : d8ebb36b-5e8e-416b-aab5-02855ae757d9
external_ids   : {}
name           : mymirror
output_port    : 2eca4ca7-e646-4905-a774-bf13d44e1a23
output_vlan    : []
select_all     : true
select_dst_port: []
select_src_port: []
select_vlan    : []
snaplen        : []
statistics     : {tx_bytes=0, tx_packets=0}
root@reda-Latitude-5490:~#

```

FIGURE 5.4 – Activation du Port-Mirroring

A- Installation du système F-DoS sur h6

Sur l'hôte **h6**, nous allons installer notre système de détection. Rappelons que le module MEI du système F-DoS fait appel au client et au serveur ARGUS. Donc en premier nous allons activer le serveur ARGUS sur l'hôte h6 à l'aide de la commande suivante :

```
% sudo argus -d -i h6-eth0 -P 561 -w /captured/today.argus
```

L'option **-i** : permet de spécifier l'interface d'écoute, qui est, dans notre cas, l'interface Ethernet 0 de h6.

On lance ensuite script "IDS.py" qui va installer le client argus, ainsi que le modèle de clustering F-Clustering.

```

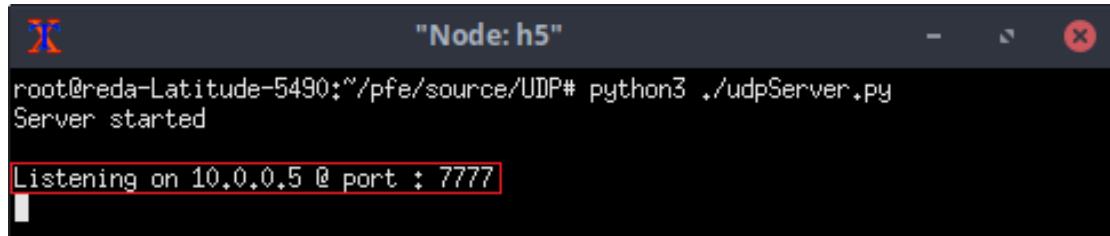
root@reda-Latitude-5490:~/pfe/source/Ids# python3 ./IDS.py -i localhost:561 -a /home/reda/pfe/sou
rce/archives/today.csv
Namespace(archive='/home/reda/pfe/source/archives/today.csv', file=None, interface='localhost:561')
F-DoS has started on localhost:561
Archiving on

```

FIGURE 5.5 – Installation du module de détection

B- Installation du serveur d'applications sur h5

Le fonctionnement de ce serveur est basique, il se met à l'écoute des requêtes des clients, qui sont juste des messages, pour les répondre avec un message "hello". Sur l'hôte h5, le script "udpServer.py" est exécuté comme montré dans la figure 5.7.

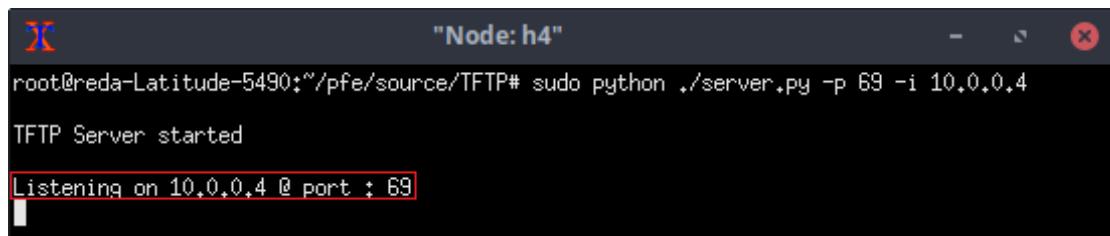


```
"Node: h5"
root@reda-Latitude-5490:~/pfe/source/UDP# python3 ./udpServer.py
Server started
Listening on 10.0.0.5 @ port : 7777
```

FIGURE 5.6 – Installation du serveur d'applications

C- Installation du serveur de fichier sur h4

Le vrai travail a été fait au niveau de cet hôte, où nous avons écrit, à partir du zéro, un script python qui lui permet de se comporter comme un serveur TFTP réel. h4 prend en charge seulement les deux fonctions, chargement et téléchargement d'un fichier. On utilisera ce serveur pour simuler le trafic TFTP qui sera capturé et analyser par notre système de détection pour l'identifier comme bénin (Téléchargement normal d'un fichier), ou bien attaque DoS. La méthode de génération des attaques réflectives va être expliquée par la suite, dans ce chapitre.



```
"Node: h4"
root@reda-Latitude-5490:~/pfe/source/TFTP# sudo python ./server.py -p 69 -i 10.0.0.4
TFTP Server started
Listening on 10.0.0.4 @ port : 69
```

FIGURE 5.7 – Installation du serveur de fichiers

D- Installation des clients

L'hôte h2 et l'hôte h1 sont, respectivement, un client du serveur d'applications (client UDP) et un client du serveur de fichiers (client TFTP). Ces deux hôtes seront aussi utilisés pour lancer des attaques réflectives contre ces deux serveurs, ainsi qu'une attaque du type UDP-Flooding.

Sur h2 on lance le script "udpClient.py" qui prend comme argument le message à envoyer au serveur.

```

root@reda-Latitude-5490:~/pfe/source/UDP# python3 ./udpClient.py -m "Salam"
Client is connecting to server 10.0.0.5:7777
Sending <Salam> to Server
Client has received <Hello Client from Server> from Server
root@reda-Latitude-5490:~/pfe/source/UDP#

```

(A) Client UDP

```

root@reda-Latitude-5490:~/pfe/source/UDP# python3 ./udpServer.py
Server started
Listening on 10.0.0.5 @ port : 7777
Server has received <Salam> from client 10.0.0.2:44987
Sending response to client

```

(B) Serveur d'applications

FIGURE 5.8 – Établissement de connexion entre le client et le serveur d'applications

D'après la figure 5.8, on peut voir qu'une connexion entre le serveur d'applications (@10.0.0.5) et le client (@10.0.0.2) a été établi sur le port 7777. Le client a envoyé le message "Salam". Le serveur lui a répondu par le message "Hello Client from Server". Ce transfert de message a fait générer un trafic UDP qui a été capturé, en temps réel, par notre système de détection et l'a classé ce flux comme bénin, qui est bien le cas.

```

F-DoS has started on localhost:561
Archiving on
Start capturing *****
10.0.0.2:47734 has sent data to 10.0.0.5:7777
Duration : 0.009239, Total Packets : 4, Packets from src 2, Packets from dst 2
Packets Length from src : 94, Packets Length from dst : 132
Flow Bytes/s : 97846.085938, Flow Packets/s : 324.710449
Source Packets/s : 227.376083, Destination Packets/s : 237.699066
Trafic : BENIGN *****

```

FIGURE 5.9 – Capture et analyse du trafic UDP

On passe à l'hôte **h1**. Le script "client.py" établit une connexion avec le serveur de fichier. À l'aide de la console tftp, **h1** peut exécuter des commandes, à distance, sur le

serveur de fichier. Dans notre cas nous avons lancé une requête "get" pour télécharger le fichier "*hello.txt*", comme indiqué dans la figure 5.10.

```
root@reda-Latitude-5490:~/pfe/source/TFTP/client# sudo python ./client.py -p 69 -i 10.0.0.4 -si 10.0.0.1 -sp 10125
Client 10.0.0.1:10125 is connected to File server on 10.0.0.4:69
tftp>> get hello.txt
Done
tftp>>
```

(A) Client TFTP

```
root@reda-Latitude-5490:~/pfe/source/TFTP# sudo python ./server.py -p 69 -i 10.0.0.4
TFTP Server started
Listening on 10.0.0.4 @ port : 69
Recieved packet
Opening read thread
Read request from: 10.0.0.1:10125, for file hello.txt
File sent
```

(B) Serveur de fichiers

FIGURE 5.10 – Établissement de connexion entre le client et le serveur de fichiers

Le transfert de fichier est réel, sur **h1** on voit le fichier téléchargé "*hello.txt*". Ce trafic de téléchargement a été aussi capturé par F-DoS et classé comme bénin, comme montré dans la figure suivante.

```
Start capturing
*****
10.0.0.1:10125 has sent data to 10.0.0.4:69
Duration : 0.000352, Total Packets : 1, Packets from src 1, Packets from dst 0
Packets Length from src : 60, Packets Length from dst : 0
Flow Bytes/s : 0.000000, Flow Packets/s : 0.000000
Source Packets/s : 0.000000, Destination Packets/s : 0.000000
TFTP request
*****
10.0.0.4:29046 has sent data to 10.0.0.1:10125
Duration : 0.051875, Total Packets : 4, Packets from src 2, Packets from dst 2
Packets Length from src : 134, Packets Length from dst : 92
Flow Bytes/s : 17426.505859, Flow Packets/s : 57.831326
Source Packets/s : 19.277109, Destination Packets/s : 201.694229
Trafic : BENIGN*****
```

Capture et analyse du trafic TFTP

5.5.3 Simulation des attaques

A- Simulation de RDoS

Rappelons que les attaques DoS réfléchis utilisent une adresse usurpée d'un hôte victime, pour envoyer des requêtes à un tiers réfléchis, qui est dans notre cas un serveur. Le serveur va répondre à la requête avec des messages réponses, mais l'hôte victime ne va pas accuser réception parce qu'il voit que ces messages ne le concernent pas et il n'a jamais lancé une telle requête, ce qui engendrera une surcharge de bande passante.

Pour simuler cette attaque, nous fixons comme victime l'hôte **h3** (@10.0.0.3). Au niveau de l'hôte **h1**, nous établissons une connexion avec le serveur de fichier avec l'adresse usurpée de **h3**, ensuite on demande le transfert du fichier "*hello.txt*".

```
"Node: h1"
root@reda-Latitude-5490:~/pfe/source/TFTP/client# sudo python ./client.py -p 69
-i 10.0.0.4 -si 10.0.0.3 -sp 20125
Client 10.0.0.3;20125 is connected to File server on 10.0.0.4;69
tftp>> get hello.txt
```

(A) Demande de téléchargement

```
"Node: h4"
root@reda-Latitude-5490:~/pfe/source/TFTP# sudo python ./server.py -p 69 -i 10.0.0.4
TFTP Server started
Listening on 10.0.0.4 @ port : 69
Received packet
Opening read thread
Read request from: 10.0.0.1:10125, for file hello.txt
File sent
Received packet
Opening read thread
Read request from: 10.0.0.1:10125, for file hello.txt
File sent
Received packet
Opening read thread
Read request from: 10.0.0.3:20125, for file hello.txt
```

(B) Envoie du fichier

FIGURE 5.11 – Simulation d'une attaque RDoS

Le serveur reçoit plusieurs fois l'envoi du fichier à l'hôte **h3**, comme montré dans figure 5.11b, car il ne reçoit aucun accusé de la part de cet hôte pour affirmer la réception du fichier.

En parallèle, avec l'exécution de l'attaque, qui prend un temps considérable, notre système F-DoS est toujours actif pour l'analyse du trafic. Le trafic généré par cette attaque a été capturé et bien classé comme attaque (voir 5.12).

```

F-DoS has started on localhost:561
Archiving on
Start capturing
*****
10.0.0.3:20125 has sent data to 10.0.0.4:69
Duration : 0.016932, Total Packets : 2, Packets from src 2, Packets from dst 0
Packets Length from src : 120, Packets Length from dst : 0
Flow Bytes/s : 28348.689453, Flow Packets/s : 59.059769
Source Packets/s : 59.059769, Destination Packets/s : 0.000000
TFTP request
*****
10.0.0.4:52583 has sent data to 10.0.0.3:20125
Duration : 0.009328, Total Packets : 2, Packets from src 2, Packets from dst 0
Packets Length from src : 134, Packets Length from dst : 0
Flow Bytes/s : 57461.406250, Flow Packets/s : 107.204117
Source Packets/s : 107.204117, Destination Packets/s : 0.000000
Trafic : ATTACK

```

FIGURE 5.12 – Détection de l'attaque RDoS

B- Simulation de UDP-Flooding

À partir de **h2**, on lance une attaque du type UDP-Flooding contre **h3** (hôte victime, @10.0.0.3) à l'aide de la commande suivante :

```
% hping3 --udp --flood 10.0.0.3 -c 10 -d 400
```

L'option :

- c : permet de spécifier le nombre de flux.
- d : permet de définir la taille des données.

```

root@reda-Latitude-5490:~/pfe/source/UDP# hping3 --udp --flood 10.0.0.3 -c 10
-d 400
HPING 10.0.0.3 (h2-eth0 10.0.0.3): udp mode set, 28 headers + 400 data bytes
hpingle in flood mode, no replies will be shown

```

FIGURE 5.13 – Simulation d'une attaque UDP-Flooding

D'après la figure 5.14, on voit que plusieurs flux ont été générés, qui est le cas d'une attaque par inondation (Flooding). F-DoS a capturé chacun de ces flux et l'a identifié comme attaque.

```

"Node: h6"
Flow Bytes/s : 2525.898193, Flow Packets/s : 0.714338
Source Packets/s : 0.714338, Destination Packets/s : 0.000000
Trafic : ATTACK
*****
10.0.0.2:1460 has sent data to 10.0.0.3:0

Duration : 2.799794, Total Packets : 3, Packets from src 3, Packets from dst 0
Packets Length from src : 1326, Packets Length from dst : 0
Flow Bytes/s : 2525.900146, Flow Packets/s : 0.714338
Source Packets/s : 0.714338, Destination Packets/s : 0.000000
Trafic : ATTACK
*****
10.0.0.2:1461 has sent data to 10.0.0.3:0

Duration : 2.799794, Total Packets : 3, Packets from src 3, Packets from dst 0
Packets Length from src : 1326, Packets Length from dst : 0
Flow Bytes/s : 2525.900146, Flow Packets/s : 0.714338
Source Packets/s : 0.714338, Destination Packets/s : 0.000000
Trafic : ATTACK
*****

```

FIGURE 5.14 – Détection de l'attaque UDP-Flooding

5.6 Conclusion

Au long de ce chapitre, nous avons présenté l'implémentation de notre solution de détection des attaques par déni de service dans un réseau SDN. Nous avons commencé par présenter l'environnement de travail suivi des outils utilisés pour la mise en oeuvre de notre solution. Nous avons aussi présenté la réalisation du module F-Clustering qui assure la fonction de détection des attaques en analysant les caractéristiques des flux en entrée. Pour voir l'efficacité de ce module nous l'avons mis sous le test, avec comme données de test le dataset "TFTP". Le module a montré des performances excellentes. Une précision de 98.6% et un taux de fausses alertes de 1.3%. Nous avons conclu, après la petite expérience faite dans la section 5.4 que les bonnes performances d'un modèle d'apprentissage depend, à grand part, du prétraitement des données d'apprentissage et

non pas du choix de l'algorithme d'apprentissage.

La section la plus importante de ce chapitre était la simulation. Nous avons commencé par décrire l'architecture du réseau SDN à simuler. L'étape suivante était la création de la topologie du réseau sur Mininet suivi de l'attribution des rôles. Nous avons défini deux hôtes qui vont jouer le rôle d'un serveur, deux hôtes clients et un hôte victime. Une connexion a été établi, entre chaque serveur et son client potentiel, pour l'échange de données. À partir d'hôte, désigné comme client, une attaque RDoS a été lancé et on a observé le comportement de notre réseau, en particulier notre système de détection, qui a réagi en temps réel aux flux générés pour l'identifier comme attaque. Une autre du type Flooding a été simulé et notre système a été capable de la détecter.

Bibliographie

- [1] Open Networking Foundation. Software-Defined Networking : The New Norm for Networks. ONF White Paper, April 13, 2012.
- [2] Open Data Center Alliance. Open Data Center Alliance Master Usage Model : Software-Defined Networking Rev. 2.0. White Paper. 2014.
- [3] OpenDaylight, <https://www.opendaylight.org/>
- [4] Ryu, <http://osrg.github.io/ryu/>
- [5] N.Gude et al. NOX : Towards an operating system for networks. SIGCOMM compute Commun Rev Vol. 38.numero 3, p.105-110, juill 2008.
- [6] Kreutz, D., et al. "Software-Defined Networking : A Comprehensive Survey." Proceedings of the IEEE, January 2015.
- [7] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodol- molky, and S. Uhlig, "Software-defined networking : A comprehensive sur-vey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2015.
- [8] M. Geva, A. Herzberg and Y. Gev, "Bandwidth Distributed Denial of Service : Attacks and Defenses," in IEEE Security & Privacy, vol. 12, no. 1, pp. 54-61, Jan.-Feb. 2014, doi : 10.1109/MSP.2013.55.
- [9] S. Kumar, "Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet," Second International Conference on Internet Monitoring and Protection (ICIMP 2007), San Jose, CA, 2007, pp. 25-25, doi : 10.1109/ICIMP.2007.42.
- [10] Tsunoda H, Ohta K, Yamamoto A, Ansari N, Waizumi Y, Nemoto Y. Detecting DR-DoS attacks by a simple response packet confirmation mechanism. Comput Commun 2008;31(14) :3299–306. doi :10.1016/j.comcom.2008.05.033.
- [11] Georgi A. Ajaeiya Nareg Adalian Imad H. Elhajj Ayman Kayssi Ali Chehab "Flow- Based Intrusion Detection System for SDN", American University of Beirut 2017
- [12] L. Breiman, "Bagging predictors," Mach. Learn., vol. 24, no. 2, pp. 123–140, 1996.
- [13] ABUBAKAR, Atiku and PRANGGONO, Bernardi. "Machine learning based intrusion detection system for software defined networks" 2017
- [14] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "knowledge discovery and data mining : Towards a unifying Framework", In Proceeding of the 2 nd International Conference on Knowledge Discovery and Data Mining, Menlo Park, California, 1996, pp. 82-88.
- [15] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., The FAIR Guiding Principles for scientific data management and stewardship, Scientific Data 3.
- [16] H. H. Jazi, H. Gonzalez, N. Stakhanova, A. A. Ghorbani, Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling, Computer Networks 121 (2017) 25–36.

- [17] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization, in : International Conference on Information Systems Security and Privacy (ICISSP), 2018, pp. 108–116
- [18] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho, Flow-based benchmark data sets for intrusion detection, in : European Conference on Cyber Warfare and Security (ECCWS), ACPI, 2017, pp. 361–369.
- [19] M. Alkasassbeh, G. Al-Naymat, A. Hassanat, M. Almseidin, Detecting Distributed Denial of Service Attacks Using Data Mining Techniques, International Journal of Advanced Computer Science and Applications (IJACSA) 7 (1) (2016) 436–445.
- [20] Canadian Institute for Cybersecurity, <https://www.unb.ca/cic/datasets/ids-2018.html>
- [21] Canadian Institute for Cybersecurity, <https://www.unb.ca/cic/datasets/ddos-2019.html>
- [22] M. Kumar, “1.7 Tbps DDoS Attack – Memcached UDP Reflections Set New Record,” Accessed on 2018-04-02. [Online]. Available : <https://thehackernews.com/2018/03/ddos-attack-memcached.html>
- [23] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, “Millions of Targets Under Attack : a Macroscopic Characterization of the DoS Ecosystem,” in 2017 ACM IMC, November 2017.
- [24] Thomas Lukaseder et al, "An SDN-based Approach For Defending Against Reflective DDoS Attacks". Institute of Distributed Systems, Ulm University, Germany.
- [25] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao, “Packetscore : a statistics-based packet filtering scheme against distributed denial-of-service attacks,” IEEE TDSC, vol. 3, no. 2, April 2006.
- [26] L. Haiqin, M.S. Kim, “Real-Time Detection of Stealthy DDoS Attacks Using Time-Series Decomposition.Communications.” Cape Town, South Africa, 23.-27.5.2010, pg. 1-6.
- [27] S. Noh, C. Lee, K. Choi, and G. Jung, “Detecting Distributed Denial of Service (DDoS) attacks through inductive learning, Lecture Notes in Computer Science, vol. 2690, pp. 286—295, 2003.
- [28] Mininet : An Instant Virtual Network on your Laptop (or other PC) – Mininet. <https://mininet.org/>
- [29] <https://fr.wikipedia.org/wiki/Scikit-learn>
- [30] <https://fr.wikipedia.org/wiki/Pandas>
- [31] Argus, <https://openargus.org/>