Détection d'anomalies de sûreté et sécurité d'un contrôle centralisé de réseau

Loïc Desgeorges

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Détection d'anomalies de sûreté et sécurité d'un contrôle centralisé de réseau

THÈSE

présentée et soutenue publiquement le 17 novembre 2022

pour l'obtention du

Doctorat de l'Université de Lorraine

en

Automatique, Traitement du signal et des images, Génie informatique

(mention génie informatique)

par

Loïc Desgeorges

Composition du jury

Président :	Didier MAQUIN	Professeur, Université de Lorraine
Rapporteurs :	Pascal BERTHOU	Maître de conférences, HDR, Université Toulouse III
	Isabel DEMONGODIN	Professeure, Université d'Aix-Marseille
	Damien MAGONI	Professeur, Université de Bordeaux
Examinateurs :	Ye-Qiong SONG	Professeur, Université de Lorraine
	Sami SOUIHI	Maître de conférences, Université Paris-Est Créteil
	Thierry DIVOUX	Professeur, Université de Lorraine (directeur de thèse)
	Jean-Philippe GEORGES	Professeur, Université de Lorraine (directeur de thèse)
Invités :	Jean-Marc FAURE	Professeur, SUPMECA Paris
	Jean-Jacques LESAGE	Professeur, École Normale Supérieure Paris-Saclay

Centre de Recherche en Automatique de Nancy — UMR 7039

Mis en page avec la classe thesul.

Remerciements

Les premières personnes que j'aimerai remercier sont mes directeurs Thierry DIVOUX et Jean-Philippe GEORGES pour ce voyage. Toutes ces discussions, scientifiques ou non, m'ont permis de mieux comprendre la carrière de chercheur, une saison de rugby et la façon de préparer un marathon (parce que oui, il y a quand même des sujets sérieux).

Il est également clair que je ne serai jamais arrivé à ce stade sans d'excellents formateurs. En master avec notamment Grégory FARAULT et Jean-Jacques LESAGE que je remercie pour leurs conseils et soutiens même lors de mes travaux de recherches. Je n'oublie pas mes échanges avec Jean-Jacques, lors de ma thèse, qui m'auront permis de mieux appréhender certains points et produire quelque chose qui me plaît et que j'aime.

Et puis il y a eu les stages qui m'ont permis de découvrir la recherche. Je commencerai par évoquer celui à Munich. Celui sans qui tout le reste aurait été différent et pour cela je vous remercie Julien PROVOST. Je me souviens de notre échange un mercredi après midi "alors si tu veux tes travaux tu peux en faire un article scientifique. Ce serait pour une conférence, WODES ? Bon alors par contre la deadline c'est vendredi." qui m'a permis de me rendre compte que ne pas dormir pendant 3 jours est possible. Merci de m'avoir fait confiance en mes travaux et d'avoir investi quelques-unes de vos nuits dans les corrections.

S'en est suivi la magnifique aventure humaine a Chatou. J'ai été extrêmement chanceux de pouvoir partager mon bureau avec Pierre-Yves PIRIOU et Thibault LEMATTRE. J'ai énormément appris à vos côtés et on a réussi à produire un très beau travail. Et surtout, le ping-pong (les sujets vraiment sérieux) avec l'organisation d'un beau tournoi !

Il me reste à remercier du fond du coeur Jean-Marc FAURE. Vous êtes l'un des (si ce n'est le) premiers à m'avoir fait confiance. Vos conseils m'ont été très précieux, je me souviens être venu dans votre bureau à Supméca pour vous partager le fait que je souhaitais devenir chercheur et en réponse à cela vous m'avez écrit sur une feuille un parcours que je pourrai suivre. J'ai pas fait très original. D'ailleurs, je tiens à mentionner que vous avez toujours su trouver les mots. Notamment au moment de me convaincre de postuler sur cette thèse. Merci beaucoup.

Ce manuscrit présente l'accomplissement scientifique de mes travaux lors de ces trois dernières années. Ce fut un voyage parfois tumultueux et dans ces moments compliqués j'avais la chance d'avoir du soutien. Il y a tout d'abord les rencontres que j'ai pu faire lors de cette vie nancéienne. Je pense notamment à l'équipe du Berthom Nancy qui m'aura permis de perfectionner mes connaissances sur la bière. Il y a également Guilain et Paul. On a commencé, travaillé et critiqué ensemble. Mention spéciale à Guilain, le support technique, qui ne comptait pas ses heures pour m'aider durant la thèse.

Il y a ceux qui sont là depuis un peu plus longtemps et sur qui j'ai toujours pu compter. Les amis, irlandais ou non, et le meilleur collègue/connaissance/partenaire de pelote. Dans le même genre, il faut mentionner la team confinement. J'ai eu la chance d'avoir une thèse full-package qui m'a fourni une personne formidable pour ces 3 ans. Un excellent soutien. Notamment lors du confinement, avec un accueil formidable, des séances d'abdos inoubliables (une surtout, mais bref c'est pas le sujet) et pleins de souvenirs.

Pour terminer, merci papa et merci maman. Alors effectivement lorsque les difficultés étaient mathématiques vous aviez un support que je qualifierai de relatif, mais dès lors qu'il fallait me soutenir vous étiez là. Plus ou moins maladroitement, mais c'est l'intention qui compte. Et au fil des années, au-delà du support vous êtes devenus des exemples. Vous me démontrez que lorsqu'on veut vraiment quelque chose il faut se battre et ne jamais abandonner.

Bien, alors maintenant est venu le moment de monter sur l'estrade et défendre ma thèse. Au moment de monter les marches, j'aurai en tête une promesse.

Table des matières

Chapitre 3
Détection d'anomalies dans un contrôle de réseau : solutions originales

Chapitre 4
Analyse par expérimentations des solutions proposées

Table des figures

Introduction générale

Internet a réussi de manière stupéfiante à modeler la façon dont nous accédons et échangeons des informations dans le monde moderne. Au cours des trois dernières décennies, l'architecture de l'Internet a prouvé sa valeur en prenant en charge une multitude d'applications distribuées et une grande variété de technologies de réseau sur lesquelles elle fonctionne actuellement. Face à de telles évolutions, le réseau doit s'adapter et les solutions classiques présentent bien des limites. C'est particulièrement vrai pour le contrôle de réseau, c'est-à-dire la façon dont les décisions sont prises sur le réseau. En particulier, les statistiques du CERT (CERT, 2021) indiquent que le nombre d'intrusions a excessivement augmenté d'année en année. Toutes les intrusions ou attaques malveillantes sur les vulnérabilités du réseau, les ordinateurs ou les systèmes d'information peuvent donner lieu à de graves crises, et violer les politiques de sécurité informatique, à savoir, la confidentialité, l'intégrité et la disponibilité. En conséquence de ces risques, des techniques de détection d'intrusion sont implémentées dans la plupart des infrastructures informatiques (Allen, Christie, Fithen, McHugh, & Pickel, 2000) même si jusqu'à présent, les menaces sur la sécurité des réseaux et d'information sont encore des sujets de recherche importants.

La littérature s'est donc beaucoup intéressée à la mise en place de systèmes de détection d'intrusion dans les réseaux (Liao, Lin, Lin, & Tung, 2013a). Les techniques de détection d'intrusion consistent à surveiller constamment le réseau afin d'y repérer des activités anormales. Ces techniques sont généralement divisées en deux catégories : celles basées sur le fait de repérer la signature d'un type d'attaque et celles basées sur la recherche d'anomalies dans le comportement nominal du système. Pour répondre aux exigences d'un système de détection des intrusions (IDS) efficace, les chercheurs ont exploré la possibilité d'utiliser des techniques de machine learning (ML) et de deep learning (DL). Ces deux approches font partie de la thématique de l'intelligence artificielle (IA) et ont pour but d'extraire et apprendre des informations utiles à partir de données volumineuses. Ces techniques ont gagné en popularité dans le domaine de la sécurité des réseaux au cours de la dernière décennie, grâce à l'accroissement de la puissance des processeurs. ML et DL sont des outils puissants pour apprendre des caractéristiques utiles du trafic réseau et prédire les activités normales et anormales sur la base des modèles appris (Tsai, Hsu, Lin, & Lin, 2009) (I. Ahmad, Namal, Ylianttila, & Gurtov, 2015).

Par ailleurs, les réseaux IP traditionnels sont complexes et difficiles à gérer. Pour exprimer les politiques de réseau de haut niveau souhaitées, les opérateurs de réseau doivent configurer séparément chaque dispositif de réseau individuellement (commutateur, routeur ...) et en plus les environnements réseau doivent supporter la dynamique des pannes et s'adapter aux changements de charge. La décentralisation classique présente des limites (Laterrasse, Chatzis, & Coutard, 1992) ce qui explique le développement d'architectures de contrôle centralisé et notamment le Software-Defined Networking (McKeown et al., 2008). D'un point de vue sécurité, cette nouvelle architecture réseau introduit de nouvelle spécificité. D'un côté la centralisation est bénéfique pour le développement de méthode de détection d'intrusion grâce à la vue globale du contrôleur

(Xie et al., 2018) (Abubakar & Pranggono, 2017) (Nanda, Zafari, DeCusatis, Wedaa, & Yang, 2016). Néanmoins ces spécificités amènent également de nouvelles menaces (Kreutz et al., 2013) principalement liées au contrôle de réseau. L'enjeu principal d'une architecture SDN concerne la sécurité et la sûreté de la couche contrôle. En effet, du fait de la centralisation du contrôle en une seule entité, la défaillance du contrôleur a pour conséquence le blocage de l'ensemble du réseau puisque plus aucune décision de contrôle ne peut être prise sur l'ensemble de l'architecture. De manière similaire, en cas d'attaque du contrôleur, un attaquant prend possession de l'ensemble du réseau et peut mettre en place le contrôle qu'il souhaite (Chica, Imbachi, & Vega, 2020) (Shin, Xu, Hong, & Gu, 2016) (P. Porras et al., 2012).

Dans la littérature, les solutions proposées afin de pallier aux menaces de sécurité et sûreté du contrôleur consistent au développement d'architectures multicontrôleurs spécifiques (Blial, Ben Mamoun, & Benaini, 2016). Traditionnellement les architectures multicontrôleurs ont été introduites pour prendre en charge des réseaux de grandes dimensions, mais cette multiplication des contrôleurs présente des bénéfices que ça soit vis-à-vis de la sûreté ou de la sécurité. En effet, que l'architecture soit distribuée de manière physique ou logique, elle permet de fournir une redondance du contrôleur en cas de panne (Fonseca, Bennesby, Mota, & Passito, 2012) (Das, Pohrmen, Maji, & Saha, 2020). Cette redondance est nécessaire en cas de défaillance du contrôleur et donc pour prendre en compte les problèmes de sûreté nous allons considérer uniquement des architectures de ce type. Vis-à-vis des problèmes de sécurité, la distribution offre également plusieurs avantages. Cela permet à chacun des contrôleurs de s'observer mutuellement et déterminer si les règles proposées par l'un des contrôleurs sont valides ou non (Qi et al., 2016). De cette façon, chaque contrôleur participe à la détection d'anomalies dans le contrôle. Plus récemment, la Blockchain a été considérée afin de sécuriser la couche contrôle (Derhab, Guerroumi, Belaoued, & Cheikhrouhou, 2021). Cependant, l'introduction d'une architecture de type multicontrôleurs amène également quelques contraintes (Bannour et al., 2017) et notamment la mise en place d'une interface de communication entre les contrôleurs appelée interface Est-Ouest. Cette interface est nécessaire pour assurer la cohérence dans le contrôle ou pour la mise en place des méthodes sécuritaires présentées précédemment. Cette interface est une menace de sécurité (Kreutz et al., 2013) : un attaquant peut propager des informations malveillantes à travers cette interface. Ainsi, différentes propositions visent à sécuriser cette interface comme par exemple (Shang et al., 2018) qui se concentre sur le développement d'un mécanisme de communication sécurisé entre les contrôleurs ou encore (Lam, Lee, Lee, & Oktian, 2015) qui propose de la chiffrer.

Cette thèse vise à proposer un contrôle sécurisé. Pour cela, on considère une architecture multicontrôleurs sans interface de communication entre eux. Cette architecture est composée d'un contrôleur en charge du contrôle de réseau et d'un second contrôleur, réduit au rôle d'observateur, qui sera en charge uniquement de la détection d'anomalie dans le contrôle. Ces anomalies pouvant être liées à des problèmes de sûreté ou bien de sécurité. La contrainte étant que l'observateur n'a pas accès aux états internes du contrôleur puisqu'on s'interdit les communications est-ouest qui sont trop dangereuses. La détection devra être basée uniquement sur l'observation de l'activité de la commande c'est-à-dire des échanges contrôleur/équipements. Vulgairement, nous allons rechercher des anomalies dans le comportement du contrôleur et non dans ce qu'il pourrait nous dire étant donné qu'il peut être manipulé. Dans un premier temps, nous définirons une spécification de l'activité de la commande et développerons un outil permettant d'obtenir une telle spécification. Elle correspond au modèle d'évolution de l'activité de la commande afin d'en déterminer le rôle du contrôleur et donc ce à quoi il doit être réactif. Puis une borne temporelle sera fixée, expérimentalement, définissant le temps de réponse attendu du contrôleur. Cela

permet de vérifier qu'il n'y a pas de défaillance ou d'attaque de type déni de service causant un retard dans le temps de réponse du contrôleur. Cette spécification permet de vérifier que le contrôleur a le comportement attendu : il réagit lorsque c'est nécessaire et ce, dans les temps fixés. C'est une condition nécessaire à la bonne conduite de la commande, mais ce n'est pas suffisant. Il faut également vérifier que les commandes envoyées sur le réseau correspondent à l'algorithme de commande en place. Nous étudierons le cas où l'algorithme est déterministe et celui où il est non déterministe. Tout d'abord, nous allons définir des propriétés structurelles définissant une décision cohérente. Puis, étant donné que cette vérification est nécessaire, mais pas suffisante, nous allons également réestimer les variables internes du contrôleur afin de vérifier que les commandes sont en accord avec ces variables.

Ainsi, cette thèse est multidisciplinaire en traitant à la fois des réseaux numériques de communication et du contrôle (et pilotage) sous l'angle de la sûreté de fonctionnement et de la sécurité en utilisant des outils de type système à évènements discrets ou algorithme de machine learning. Elle s'inscrit (et a été financée) dans la cadre du projet DigiTrust Lorraine Université d'Excellence (référence ANR-15-IDEX-04-LUE). Plus particulièrement, cette thèse va se concentrer sur le concept émergent de l'observabilité de réseaux, concept bien connu de la théorie de l'automatique, mais peu dans les réseaux.

Cette thèse est structurée en quatre parties. Le premier chapitre vise à contextualiser le sujet de cette thèse. Cela débute par un rapide état de l'art sur les techniques de détection d'intrusion dans les réseaux avec une présentation des différentes approches puis l'intérêt des algorithmes de Machine Learning pour cette thématique est présenté. Ensuite, le modèle d'architecture réseau de type SDN est présenté avec une description de chacune des couches ainsi que les bénéfices qu'apporte une telle centralisation dans les réseaux. Les risques liés à cette architecture sont introduits avec un zoom sur les menaces du cœur de réseau qu'est le contrôle. Un état de l'art sur les solutions proposées dans la littérature face à de telles menaces permettra de définir les réponses possibles et l'approche de nos travaux.

Dans un second chapitre, l'architecture de contrôle est ainsi présentée et le problème sous-jacent de détection est défini. Ce problème d'observabilité est formalisé avec notamment une formalisation de l'architecture de contrôle au niveau de l'interface Sud et plus particulièrement des primitives du protocole OpenFlow. Un état de l'art des méthodes de détection d'anomalies est présenté afin de positionner ces travaux. Cela permet d'introduire le principe de la méthode de détection qui va être mis en place avec une démonstration sur un exemple simple.

Dans un troisième chapitre, la contribution scientifique de cette thèse est introduite. Tout d'abord, un ensemble de conditions nécessaires à la bonne conduite du contrôle sont introduites. Ces conditions sont liées au comportement temporel du contrôleur puis à des propriétés structurelles que doivent vérifier ces décisions afin d'être cohérentes. Ensuite, un algorithme de détection est présenté et mis en situation dans le cas d'un contrôle déterministe. Enfin, cette méthode d'évaluation des états internes est étendue au cas d'un contrôle non déterministe avec l'introduction d'une méthode de détection multicritères. Deux critères sont proposés en lien avec l'application considérée et plusieurs solutions d'apprentissage sont introduites.

Enfin, ces propositions sont mises en pratiques sur divers cas d'étude dans le quatrième chapitre. Cela permet de démontrer la faisabilité des méthodes introduites et donc la détection des diverses menaces de la couche contrôle. Différents cas de défaillance, de compromission et d'étouffement du contrôleur sont mis en place. On verra que les méthodes proposées dans cette thèse

permettent de détecter les menaces, mais que les performances évoluent selon la configuration souhaitée de l'observateur.

Cette thèse se conclut sur les limites et les perspectives de ces travaux permettant leurs enrichissements et leurs adaptations aux évolutions des réseaux du futur.

Chapitre 1

Sécurité et sûreté d'un contrôle de réseau SDN

Ce chapitre vise à contextualiser le sujet de cette thèse. Tout d'abord, le contexte sur le contrôle centralisé de réseau est présenté. Puis, une architecture de contrôle de réseau centralisé est introduite : le paradigme Software-Defined Networking (SDN). Cette architecture a été popularisée par l'Open Networking Foundation en 2011 (McKeown et al., 2008) dans le but de proposer une plus grande agilité dans le déploiement et le fonctionnement des infrastructures réseau de par la centralisation du contrôle. Les différentes menaces liées à cette architecture sont développées avant de faire un zoom sur les menaces liées au contrôle de réseau qui est le sujet de cette thèse. Enfin, un état de l'art des solutions liées à ces menaces est présenté ainsi qu'un positionnement de nos travaux vis-à-vis de ces propositions. Pour terminer, une conclusion clôt ce chapitre.

1.1 Contexte

Le routage est le mécanisme par lequel des chemins sont sélectionnés dans un réseau pour acheminer les données d'un expéditeur jusqu'à un ou plusieurs destinataires. Le routage est une tâche exécutée dans de nombreux réseaux, tels que le réseau téléphonique, les réseaux de données électroniques comme Internet, et les réseaux de transports. Les routeurs sont des équipements transmettant les paquets de données d'un réseau à un autre. Les appareils utilisent des tables de routage internes ou des cartes comme outils de navigation afin de déterminer le meilleur itinéraire pour transférer des données entre les réseaux. Dans le routage, le plan de contrôle détermine le chemin à utiliser pour transmettre un ensemble de flux de données. Le plan de contrôle est responsable de l'alimentation de la table de routage, de la table de transfert et donc de l'activation des fonctions du plan de données. C'est ici que le routeur prend sa décision. En une seule ligne, on peut dire qu'il est responsable de la manière dont les paquets doivent être acheminés. Le plan de données désigne toutes les fonctions et tous les processus qui transmettent les paquets/trames d'une interface à l'autre en fonction de la logique définie par le plan de contrôle. La table de routage, la table de transfert et la logique de routage constituent la fonction du plan de données. Les paquets du plan de données passent par le routeur et l'entrée et la sortie des trames sont effectuées sur la base de la logique du plan de contrôle. En une seule ligne, on peut dire qu'il est responsable du déplacement des paquets de la source à la destination.

Classiquement ces deux plans sont mis en place de manière distribuée et chaque routeur est autonome dans sa gestion. En conséquence, chaque routeur nécessitait également un traitement

individuel. L'expérience a montré que la grande complexité qui sous-tend la conception et la configuration des réseaux d'entreprise entraîne généralement une intervention manuelle importante lors de la gestion des réseaux. Bien qu'il soit difficile de trouver des données concrètes impliquant la complexité dans les pannes de réseau, des preuves anecdotiques et des entretiens avec des opérateurs suggèrent que les réseaux plus complexes sont plus sujets aux pannes et sont surtout difficiles à mettre à niveau et à gérer (Benson, Akella, & Maltz, 2009). La gestion des coûts et le manque d'efficacité sont des défis majeurs d'un système de contrôle d'accès décentralisé. Des opérations normalement simples, comme la modification des droits d'accès, peuvent s'avérer complexes, longues et coûteuses. Concernant la gestion des risques, il est fastidieux de vérifier que tous les systèmes de contrôle d'accès décentralisés sont à jour, et conformes à la politique de sécurité en place (celle d'une entreprise par exemple). Sans parler des lois et règlements pouvant être spécifiques à seulement certaines parties du réseau (un site précis d'une entreprise étant soumis à une politique plus stricte par exemple). L'administration, le contrôle et la conformité sont des aspects essentiels du contrôle d'accès, et leurs divergences peuvent vite devenir des obstacles à la bonne gestion du réseau.

Ainsi, les réseaux informatiques traditionnels sont complexes et très difficiles à gérer (Benson et al., 2009). Pour gérer le plan de gestion, c'est-à-dire mettre en place les politiques souhaitées, les opérateurs de réseau doivent configurer, un par un, chaque équipement, soit manuellement, soit à l'aide de scripts de bas niveau.

La principale limite des architectures distribuées classiques est que même les opérations simples, telles que l'introduction d'une nouvelle configuration sur le réseau, nécessitent l'intervention d'un administrateur et l'arrêt des équipements pour la configuration manuelle sur chaque équipement. Par conséquent, les performances des réseaux gérés à l'aide de ces paradigmes sont fortement limitées par l'expertise des opérateurs humains (Samaan & Karmouch, 2009). En plus de la complexité de la configuration, les environnements réseau doivent supporter la dynamique des pannes et s'adapter aux changements de charge. Chacune de ces évolutions nécessite un traitement de chaque équipement d'interconnexion du réseau. L'application des politiques requises dans un environnement aussi dynamique est un véritable défi. Le plan de contrôle (qui décide de la manière de gérer le trafic réseau) et le plan de données (qui achemine le trafic en fonction des décisions prises par le plan de contrôle) sont regroupés dans les équipements d'interconnexion. Il s'agit d'un obstacle fondamental qui a pu conduire à la lenteur de l'innovation de l'infrastructure de mise en réseau (Ramos, Kreutz, & Verissimo, 2015). Tout cela a réduit la flexibilité, puisqu'en cas d'évolution, chaque équipement doit être modifié un à un, et l'évolution des infrastructures réseau. Un exemple des difficultés est la transition d'IPv4 à IPv6, entamée il y a plus de dix ans et encore largement inachevée (Ghodsi et al., 2011). En raison de l'inertie des réseaux IP actuels, un nouveau protocole de routage peut prendre 5 à 10 ans pour être entièrement conçu, évalué et déployé (Ghodsi et al., 2011).

De plus, la croissance sans précédent des demandes et du trafic de données, l'émergence de la virtualisation des réseaux ainsi que l'utilisation sans cesse croissante d'équipements mobiles dans l'environnement des réseaux modernes ont mis en évidence des problèmes majeurs qui sont fondamentalement inhérents à l'architecture distribuée conventionnelle d'Internet. Cela a rendu la tâche de gérer et de contrôler les informations provenant d'un nombre croissant d'équipements connectés de plus en plus complexes et spécialisés. En outre, dans l'architecture traditionnelle où la logique de contrôle est purement distribuée et localisée, la résolution d'un problème de réseau spécifique ou l'ajustement d'une politique de réseau particulière nécessite d'agir simultanément sur plusieurs équipements afin d'en modifier manuellement leur configuration. Dans ce contexte, la croissance actuelle des équipements d'interconnexion et des données a exacerbé les problèmes d'évolution en rendant ces interventions humaines et ces opérations réseau plus difficiles et plus

sujettes aux erreurs (Benson et al., 2009) (Bannour et al., 2017).

Par ailleurs, il est devenu particulièrement difficile pour les réseaux actuels de fournir le niveau requis de qualité de service (QoS). En effet, la qualité de service repose sur des paramètres de performance, comme la bande passante et la latence. Et, satisfaire ce nombre croissant de paramètres de performance est une tâche d'optimisation complexe qui peut être traitée comme un problème NP-complet. En outre, les opérateurs de réseaux réalisent de plus en plus que l'expérience globale de l'utilisateur final et sa perception subjective des services fournis sont aussi importantes que les mécanismes basés sur la QoS. Par conséquent, les tendances actuelles en matière de gestion de réseau se dirigent vers ce nouveau concept communément appelé Qualité d'Expérience (QoE) pour représenter la qualité globale d'un service de réseau du point de vue de l'utilisateur final. La QoE introduit des exigences supplémentaires axées sur l'utilisateur tandis que la QoS ne concerne que l'équipement du réseau. Ainsi, on peut constater un énorme écart entre les progrès réalisés dans les technologies informatiques et logicielles et entre la stagnation des types d'infrastructure réseau sous-jacente, non évolutive et difficile à gérer. Cet écart permet de mettre en évidence la nécessité d'une plateforme de réseau automatisée qui facilite les opérations de réseau (Bannour et al., 2017). C'est dans ce contexte que plusieurs stratégies de recherche ont été proposées pour intégrer des approches automatiques et adaptatives dans l'infrastructure actuelle afin de relever les défis de l'évolution, de la fiabilité et de la disponibilité du trafic en temps réel, et donc de garantir la qualité d'expérience de l'utilisateur.

Ainsi, des alternatives radicales soutiennent qu'une toute nouvelle architecture de réseau devrait être construite à partir de zéro en rompant avec l'architecture de réseau conventionnelle et en apportant des changements fondamentaux pour répondre aux exigences actuelles et futures.que les performances évoluent selon la configuration souhaitée de l'observateur.

C'est dans ce contexte que de premiers travaux d'automatisation des équipements réseau ont été proposés (Calvert, 2006). En effet, en réponse à ces limitations, certains chercheurs en réseau ont adopté une autre approche pour ouvrir le contrôle des réseaux, en se basant sur l'analogie avec la facilité relative de reprogrammer un PC autonome. Cette notion d'informatique autonome (Kephart & Chess, 2003) désigne des entités informatiques capables d'effectuer des opérations par elles-mêmes afin d'atteindre un ensemble d'objectifs prédéfinis par leurs administrateurs/utilisateurs et s'inspire des systèmes nerveux humains qui régissent les fonctionnalités vitales de manière autonome (Samaan & Karmouch, 2009). La hiérarchie des composants d'un nœud est illustrée à la Fig. 1.1. L'architecture d'un réseau actif se compose d'un ensemble de nœuds. Chaque nœud exécute un ou plusieurs environnements d'exécution (EE) (de la même façon qu'un Shell Unix qui peut exécuter des paquets actifs). Les utilisateurs invoquent des applications actives (AA), qui fournissent du code pour programmer un EE afin de mettre en œuvre un service de bout en bout. Les environnements d'exécution ont accès aux ressources du nœud (bande passante utilisée et disponible, espace de stockage) par l'intermédiaire d'un système d'exploitation de nœud (NodeOS), qui est chargé de gérer/partager ces ressources entre les EE au niveau du nœud. Un réseau actif offre la possibilité d'effectuer rapidement des changements hautement adaptés, et ce, « en temps réel » lors d'opérations de réseau.

Les efforts ultérieurs se sont concentrés principalement, et presque exclusivement, sur le contrôle et la gestion de la configuration. L'idée est de mettre en place une distinction et une séparation claires entre les fonctions des plans de contrôle et de données puisque ce couplage est un frein à l'innovation du plan de contrôle. Cette barrière devient problématique au début des années 2000 avec l'augmentation des volumes de trafic et l'importance accrue accordée aux performances des réseaux. En effet, dans les réseaux traditionnels, les routeurs et les commutateurs intègrent à la fois le plan de contrôle et le plan de données. Ce couplage rend extrêmement difficiles

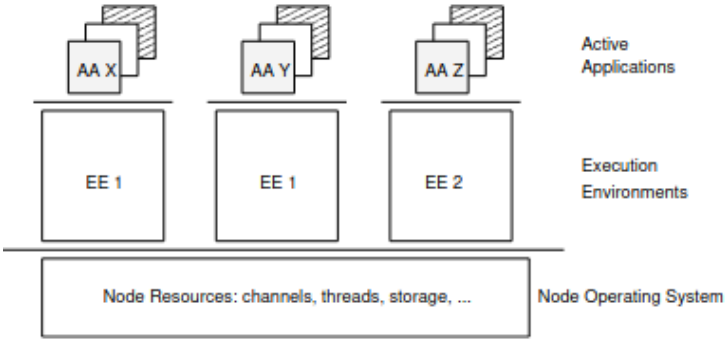


FIGURE 1.1 – Architecture d'un réseau actif (Calvert, 2006)

diverses tâches de gestion de réseau, comme la correction des problèmes de configuration et de reconfiguration. Pour répondre à ces défis, divers efforts visant à séparer les plans de contrôle et de données ont commencé à voir le jour (Feamster, Rexford, & Zegura, 2013).

En 2002, ForCES est introduit (Halpern et al., 2010) et propose une centralisation distinguant notamment les équipements nommés "Forwarding Elements" (FE) et ceux nommés "Control Elements" (CE). L'approche initiale intitulée ForCes prévoyait aussi la centralisation du plan de contrôle, mais portait surtout l'idée de séparer plan de contrôle et plan de données, en favorisant ainsi l'indépendance entre les deux plans. Une architecture ForCES est représentée en Fig. 1.2 (Haleplidis et al., 2015).

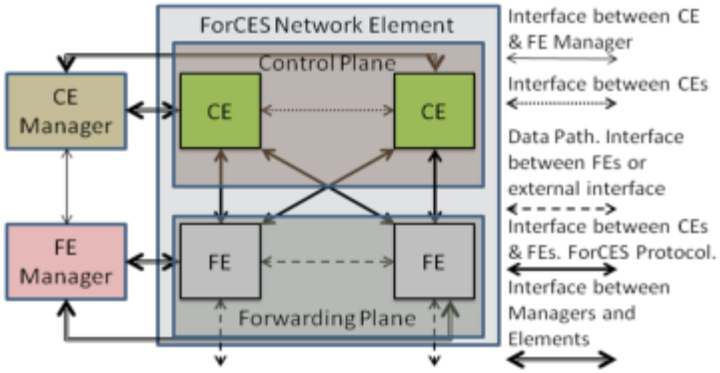


FIGURE 1.2 – Architecture ForCES (Haleplidis et al., 2015)

Dans le cadre de ForCES, un élément du réseau (NE) est composé d'éléments de contrôle (CE) et d'éléments d'acheminement (FE). Pour qu'un CE puisse contrôler n'importe quel FE, le groupe de travail a décidé de développer un modèle d'abstraction pour les FE. De la même façon, pour faire face aux différentes fonctionnalités du plan de données, le groupe de travail ForCES a normalisé un langage de modélisation qui permet aux développeurs de définir leurs propres modèles d'abstraction des FE. Pour faire face à l'extensibilité que le modèle ForCES permet, le groupe de travail a normalisé un protocole qui permet aux CEs et FEs de communiquer et aux CEs de contrôler et de gérer tout autre modèle ForCES (Haleplidis et al., 2015).

Les CE sont des entités dédiées au plan de contrôle et utilisent le protocole ForCES pour indiquer à un ou plusieurs FE comment traiter les paquets. Les protocoles de routage tels que RIP, OSPF et BGP sont des exemples de fonctions de contrôle qui peuvent être mises en œuvre

par un CE. Les paquets qui ne peuvent pas être traités par le FE lui-même sont redirigés vers le CE pour être qu'il détermine et indique au FE quoi en faire. (Haleplidis et al., 2015). Les FE, quant à eux, sont des entités du plan de données. Ils assurent le traitement et la gestion des paquets selon les instructions/contrôles d'un ou plusieurs CE (Haleplidis et al., 2015). C'est-à-dire qu'ils s'occupent d'acheminer les paquets selon les instructions des CE. Par ailleurs, comme l'illustre la figure 2 un modèle ForCES comprend deux autres entités que sont le gestionnaire de CE (CEM) et le gestionnaire de FE (FEM). Le CEM et le FEM déterminent quels FE doivent s'associer à quels CE et leur fournissent les détails d'association nécessaires, tels que les IP des éléments, avant toute communication entre CE et FE.

Le vif intérêt manifesté depuis de nombreuses années pour permettre la facilitation de la programmabilité des réseaux ainsi que la dissociation du plan de données et du plan de contrôle a abouti à la création du paradigme Software-Defined Networking (SDN) (McKeown et al., 2008). Le SDN est une architecture de réseau dans laquelle le comportement du réseau est défini par les applications au moyen d'interfaces ouvertes et de couches d'abstraction. L'une des principales caractéristiques de conception du SDN est le découplage du plan d'acheminement et du plan de contrôle. Bien que le SDN ne soit ni la premier ni la seule approche à préconiser une telle séparation, il vise à en tirer le meilleur parti des précédentes. Nous allons développer la présentation de cette architecture ci-après.

1.2 Architecture Software-Defined Networking

C'est donc dans cette idée que l'Internet Engineering Task Force (IETF) voulait centraliser la partie contrôle. Cette centralisation a ainsi été popularisée plus tard par l'Open Networking Foundation en 2011 avec l'introduction du Software-Defined Networking (SDN) (McKeown et al., 2008) représenté à droite de la Fig. 1.3.

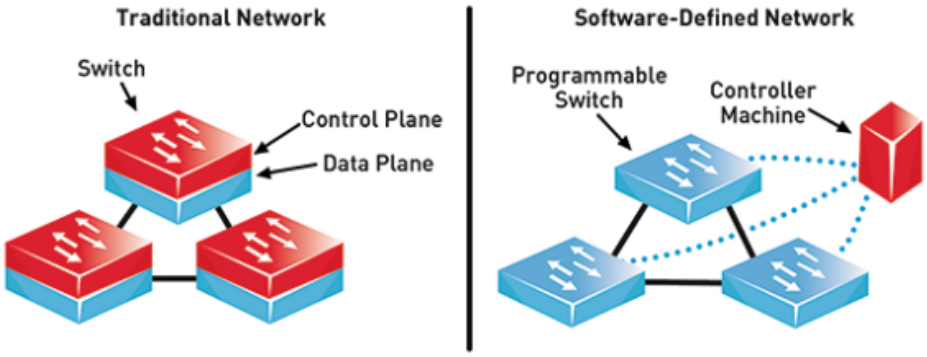


FIGURE 1.3 – Centralisation du contrôle (Hosny et al., 2020).

Cette architecture s'est retrouvée dans un premier temps au cœur de plus en plus d'études académiques, puis dans un second temps utilisée dans l'industrie, chez Google par exemple (Mandal, 2015). L'idée principale du SDN est de découpler le plan de contrôle du plan de données afin de permettre une gestion souple et efficace du réseau par le biais de programmes logiciels. Plus précisément, les équipements (par exemple, les commutateurs et les routeurs) présents dans l'infrastructure sont responsables du transfert des paquets, sur la base de règles installées par le contrôleur. Le contrôleur supervise l'ensemble des plans (données et contrôle) ce qui permet un

fonctionnement souple dans le but de mettre en œuvre diverses applications de réseau telles que du routage, des pare-feux, etc .... Cette souplesse vient du fait qu'au lieu de programmer manuellement plusieurs équipements matériels spécifiques à un fabricant, les développeurs peuvent contrôler le flux de trafic sur un réseau simplement *via* la programmation d'un seul contrôleur. En outre, le SDN permet une centralisation de la logique du contrôle en retour avec de meilleures décisions basées sur une vue globale et omnisciente du réseau.

1.2.1 Définition

L'organisation Open Networking Foundation (ONF) (Foundation, 2011) est un consortium à but non lucratif dédié au développement, à la normalisation et à la promotion du SDN. L'ONF a fourni la définition la plus explicite du SDN comme suit :

Software-Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable. (Alto, April 2012)

Selon cette définition, le SDN se définit principalement par deux caractéristiques : le découplage des plans de contrôle et de données, ainsi que la programmabilité sur le plan de contrôle. En sus, SDN repose sur quatre piliers (Kreutz et al., 2014) (Jammal, Singh, Shami, Asal, & Li, 2014) :

1. Les plans de contrôle et de données sont divisés et séparés. L'objectif étant de retirer la fonctionnalité de contrôle des commutateurs du réseau qui deviendront de simples éléments de transmission (de paquets).
2. La logique de contrôle est déplacée vers une entité externe, appelée contrôleur SDN ou système d'exploitation de réseau (NOS). Le NOS est une plateforme logicielle qui fonctionne sur une technologie de serveur de commodités et fournit les ressources et les abstractions essentielles pour faciliter la programmation des commutateurs sur la base d'une vue abstraite et logiquement centralisée du réseau. La particularité du SDN induite de ces deux premiers points est représentée Fig. 1.3.
3. Les décisions d'acheminement sont généralement basées sur le flux, et non sur la seule destination. Un flux est défini de manière générale par un ensemble de valeurs de champs de paquets agissants comme un critère de correspondance (filtre) auquel on associera un ensemble d'actions (instructions). Dans le contexte du protocole OpenFlow (ONF, Juin, 2012) servant à la mise en place du contrôle, un flux est ainsi une séquence de paquets entre une source et une destination. Tous les paquets d'un flux reçoivent des politiques de service identiques au niveau des commutateurs (Gude et al., 2008). La considération de flux permet une grande flexibilité eu égard au seul paquet, limitée uniquement par les capacités des tables de flux implémentées (McKeown et al., 2008).
4. Le réseau est programmable par le biais d'applications logicielles fonctionnant au-dessus du contrôleur SDN et interagissant avec les équipements d'interconnexion sous-jacents du plan de données.

1.2.2 Architecture

Comme déjà mentionnée, l'"intelligence" est retirée des commutateurs pour passer à un système de contrôle dont la logique est centralisée. En conséquence, une architecture SDN, comme représentée en Fig. 1.4, peut être découpée en différentes couches et chacune de ces couches a ses propres fonctions. Nous allons développer le rôle de chacune des couches telles qu'introduites par (Braun & Menth, 2014).

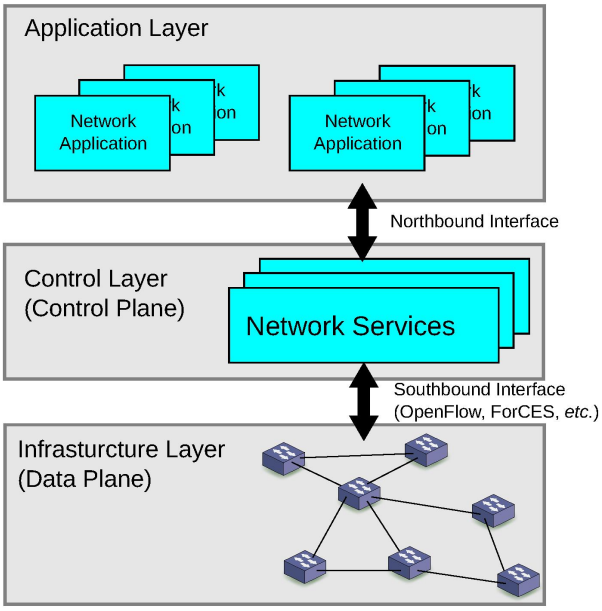


FIGURE 1.4 – Architecture SDN (Braun & Menth, 2014).

1.2.2.1 Couche infrastructure

La couche infrastructure est composée par les équipements réseau responsables du transfert de paquets et la couche est uniquement capable d'appliquer des règles envoyées par le contrôleur. La transmission de ces règles se fait par l'interface sud, c'est-à-dire l'interface de communication entre la couche contrôle et la couche infrastructure. Aujourd'hui, le protocole OpenFlow, standardisé par l'ONF (ONF, Juin, 2012), est la norme de l'interface sud la plus largement acceptée et déployée dans les architectures SDN. Elle fournit une spécification commune pour le canal de communication entre les équipements d'interconnexion (les commutateurs et le(s) contrôleur(s)).

Dans le cadre du protocole OpenFlow, les commutateurs stockent les règles dans une table flux où les règles sont constituées de trois parties : 1) une règle de correspondance (la partie "match" ou "rule" correspondant à des propriétés du flux tel que l'adresse MAC source ou celle de destination ou bien encore le port TCP source), 2) des actions à exécuter sur les paquets correspondants (par exemple le(s) port(s) de sortie sur le(s)quel(s) les paquets entrants doivent être commutés, l'encapsulation et la (re)transmission au contrôleur ou bien l'écartement de la requête), et 3) des compteurs qui correspondent à des statistiques relatives aux flux et actions. Ce modèle, représenté en Fig. 1.5, est actuellement la conception la plus répandue des commutateurs SDN et c'est celui que nous utiliserons.

1.2.2.2 Couche contrôle

Il y a différentes façons d'organiser et de penser cette couche.

La couche de contrôle SDN est communément appelée système d'exploitation du réseau (NOS) car elle prend en charge la logique de contrôle du réseau et fournit à la couche application une vue abstraite du réseau global, mais qui contient suffisamment d'informations pour spécifier des politiques tout en masquant tous les détails de mise en œuvre. Dans le plan du contrôleur, un programme logiciel est placé au sommet de l'architecture et est séparé des commutateurs comme le montre la Fig. 1.4. Le plan de contrôle manipule la table d'acheminement pour chaque demande

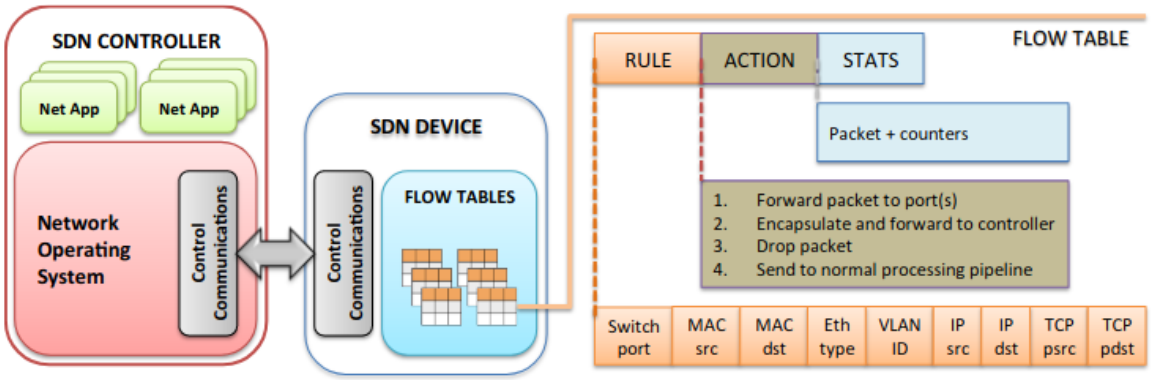


FIGURE 1.5 – Fonctionnement d'un switch SDN (Kreutz et al., 2014)

de commutateur sur la base de l'en-tête de la requête des commutateurs sous forme de `Packet_In` et envoie la réponse via le message `Packet_Out` ou `Flow_Mod` et suit toutes les demandes de la troisième couche présentée ci-dessous. Les commandes de type `Packet_Out` sont à usage unique dans la mesure où elles ne sont pas conservées par les commutateurs contrairement aux `Flow_Mod` qui sont utilisées par les commutateurs afin de remplir la table de flux. Dans cet objectif, le contrôleur doit donc compléter la partie `match` (à partir des informations sur le flux contenues dans la requête) et l'action correspondant à la décision du contrôle. En effet, ces commandes sont utilisées par le contrôleur afin de manipuler les tables de flux des commutateurs, comme représenté en Fig. 1.5, et cela permet de mettre en œuvre le plan de contrôle.

1.2.2.3 Couche Application

Par-dessus la couche contrôle, on retrouve la couche application qui interagit avec la couche contrôle *via* une interface appelée interface nord. Les interfaces nord et sud sont deux abstractions clés de l'écosystème SDN. Comme déjà mentionnée, l'interface sud a déjà plusieurs standardisations (dont une très populaire, OpenFlow), mais celle de l'interface nord est encore une question ouverte. Certains contrôleurs, tels que Floodlight, NOX, Onix et OpenDaylight, proposent et définissent leurs propres API de liaison nord. Cependant, chacun d'entre eux conserve sa propre définition.

Les applications accèdent à une vue globale du réseau *via* l'interface nord et peuvent par programmation implanter des stratégies pour manipuler les réseaux physiques sous-jacents en utilisant un langage de haut niveau fourni par la couche de contrôle. Les applications réseau peuvent ainsi être considérées comme les « cerveaux du réseau » (Kreutz et al., 2014). Elles concourent à la mise œuvre la logique de contrôle (qui sera traduite en commandes à installer dans les commutateurs). Au-delà de la sélection des fonctions réseau classiques comme le routage (sélection d'un algorithme ou encore d'une métrique), les applications correspondent à des fonctions métiers. Malgré la grande variété des cas d'utilisation, la plupart des applications SDN peuvent être regroupées dans l'une des cinq catégories suivantes : ingénierie du trafic, mobilité et sans-fil, mesure et surveillance, sécurité et fiabilité et mise en réseau des centres de données.

De manière générale, ces applications sont internes au contrôleur (et les échanges se font par API interne au contrôleur).

1.2.3 Fonctionnement basique d'une architecture SDN

L'objectif de cette section est de montrer sur un exemple simple comment fonctionne une architecture SDN. On s'intéresse à l'architecture présentée en Fig. 1.6. Elle est composée de deux équipements connectés à un contrôleur dont le but est de mettre en place de la commutation de niveau deux.

Dans notre exemple, PC1 va émettre un flux en direction de PC2 sous la forme d'un ping. Le processus physique d'échange des paquets, notamment au niveau de l'interface sud, est représenté en Fig. 1.6.

Le processus d'échange et de retransmission des paquets est décrit en Fig. 1.6.

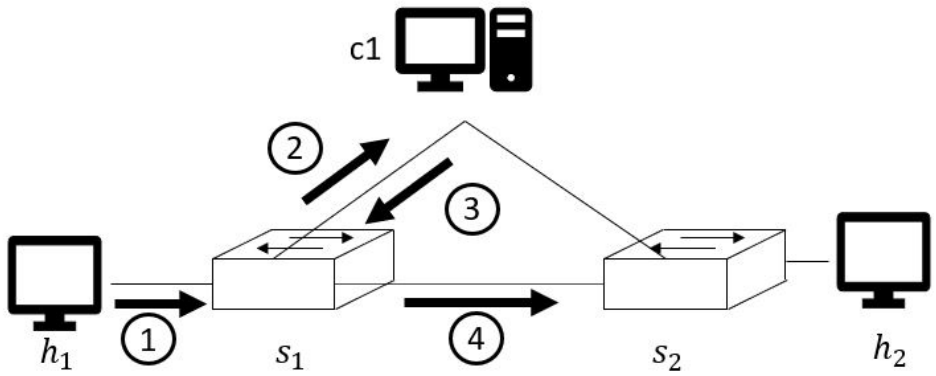


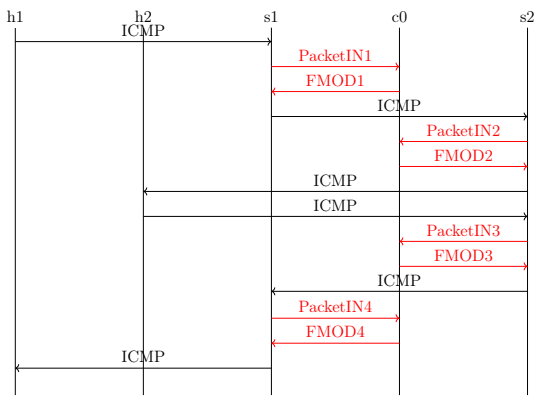
FIGURE 1.6 – Les échanges au sein de l'architecture réseau lors de la transmission d'un flux.

1. PC1 transmet le paquet au switch $sw1$.
2. Lorsque le switch $sw1$ reçoit le paquet, il vérifie le contenu de sa table de flux. Ici, on suppose que cette table est vide. Le comportement donc par défaut du switch est de transmettre le paquet au contrôleur. La requête est donc transmise en utilisant le protocole OpenFlow sous la forme d'un $Packet\_In$.
   Étant donné que le switch n'a pas de consigne sur quoi faire pour ce type de message alors il doit demander au contrôleur.
3. Le contrôleur reçoit la requête et à partir de l'image qu'il a du réseau ainsi que de son algorithme de commande (Dijkstra par exemple) va décider d'une action. Cette action est transmise au switch par le protocole OpenFlow mais cette fois sous la forme d'un `Flow_Mod`. Ce type de paquet permet d'écrire dans la table de flux du switch. Et permets donc au switch de stocker l'information.
   On peut y voir une partie "of_match" visant à définir le flux qui devra être traité par cette commande, ici c'est bien pour 01 en direction de 02. L'action à appliquer est définie dans "of_action_output".
4. À la réception de la commande, le switch applique l'action ordonnée et ici transmet au switch $sw2$.
5. Le même processus est répété, ce qui va permettre au paquet d'arriver au niveau du PC2.

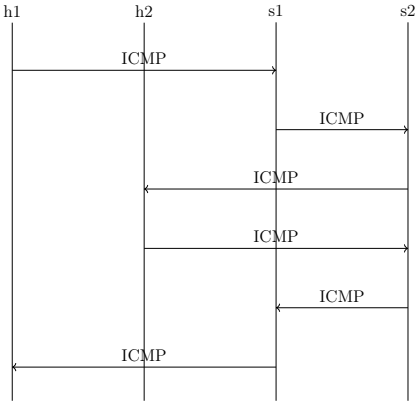
Plus en détail nous allons établir le diagramme de séquence de cet échange de paquets lors du ping selon que l'on soit dans une architecture de type SDN ou bien une architecture de réseau classique. Le diagramme est donné en Fig. 1.7, en Fig. 1.7a on retrouve le diagramme de séquence pour une architecture SDN, où les échanges OpenFlow sont en rouge, alors qu'en Fig. 1.7b est

représenté le ping pour une architecture classique. Je précise qu'on ne traite que l'échange ICMP classique (on suppose que la table ARP est déjà complétée).

La différence majeure entre ces deux séquences correspond à l'échange avec le contrôleur. En effet, dans le cas d'une architecture SDN les commutateurs n'ont pas la faculté de prendre des décisions et doivent demander au contrôleur. Cela se retrouve dans les échanges OpenFlow avec les requêtes des commutateurs sous forme de `Packet_In` et les instructions dictées par le contrôleur sous forme de `Flow_Mod` ($FMOD$). C'est cette commande qui permet au commutateur de déterminer quoi faire du paquet. Ces échanges rouges représentés en Fig. 1.7a correspondent au processus représenté en Fig. 1.6. Alors que dans une architecture classique, lorsque le commutateur reçoit le flux il détermine lui-même vers où commuter le paquet, par exemple $s1$ transmet directement à $s2$ comme on peut le voir dans Fig. 1.7b. Cette décision est prise par le commutateur directement puisque dans une architecture classique le contrôle est distribué et donc $s1$ applique l'algorithme de contrôle.



(a) Cas d'une architecture SDN



(b) Cas d'une architecture réseau classique

FIGURE 1.7 – Diagramme de séquence des échanges lors d'un ping de PC1 vers PC2

1.2.4 Avantages du SDN

La centralisation offerte par le SDN permet une plus grande évolutivité et liberté dans la programmation d'un contrôle de réseau (Xia, Wen, Foh, Niyato, & Xie, 2014). Cette caractéristique apporte des avantages comme une simplification de la configuration, une amélioration des performances et une innovation encouragée dans l'architecture et l'exploitation des réseaux, (Kirkpatrick, 2013) quel que soit le domaine d'application (Bertaux et al., 2015) (Al Hasrouty, Lamali, Magoni, & Murphy, 2017). Les possibilités de contrôle offertes par le SDN sont également très variées et peuvent inclure non seulement l'acheminement des paquets au niveau de la commutation pour du routage. De plus, cette vision globale du contrôle sur les équipements d'interconnexion offre la capacité d'acquérir l'état instantané du réseau et permet donc un contrôle centralisé en temps réel d'un réseau basé à la fois sur l'état instantané du réseau et sur les politiques définies par l'utilisateur et les applications. Par ailleurs, l'architecture SDN a été introduite afin de développer des plateformes pratiques pour l'expérimentation de nouvelles techniques et encourage de nouvelles conceptions de réseau (McKeown et al., 2008).

— Dans le contexte de gestion des réseaux, leur configuration est un enjeu délicat. En effet, à l'ajout de nouveaux équipements, il est nécessaire de mettre en place des configurations

	SDN	Réseau classique
Caractéristiques	Séparation du plan de données et de contrôle	Contrôle distribué
Configuration	Automatique avec une vision globale	Configuration manuelle de chaque commutateur
Contrôle	Algorithmes centralisés	Décision locale
Innovation	Facilité d'implémentation des propositions	Difficulté liée au matériel (hardware)

TABLE 1.1 – Comparaison entre une architecture SDN et un réseau classique

adaptées afin d'assurer le fonctionnement cohérent du réseau dans son ensemble. Sur une architecture classique, cette configuration est fastidieuse puisque chaque composant doit être reconfiguré un à un. Cette difficulté est la même lorsqu'un dépannage est nécessaire sur le réseau (à la suite de la défaillance d'un équipement par exemple). C'est la raison pour laquelle, la reconfiguration automatique et dynamique d'un réseau est un défi. L'architecture SDN permet de répondre à ces problématiques grâce à la gestion centralisée qui permet de configurer et optimiser dynamiquement un réseau entier depuis un seul et unique équipement : le contrôleur (Kirkpatrick, 2013).

— L'optimisation des performances d'un réseau classique est une question difficile du fait de l'hétérogénéité des technologies et les informations uniquement locales des équipements. La vision globale et abstraite du contrôleur SDN permet à tout instant d'avoir une vision de l'état du réseau et donc une optimisation des performances par des algorithmes centralisés. Dans la littérature, les bénéfices du SDN ont été utilisés afin de faire des propositions pour répondre à des problèmes de réseau classiques tels que l'ordonnancement du trafic de données (Qin, Dai, Huang, & Xu, 2015), le contrôle de la congestion (Hafeez, Ahmed, Ahmed, & Malik, 2017) ou le routage à équilibrage de charge (Zakia & Yedder, 2017).

— La facilité de configuration du réseau dans une architecture SDN permet d'encourager l'innovation en fournissant une plateforme réseau programmable pour mettre en œuvre, expérimenter et déployer de nouvelles idées et de nouvelles applications (McKeown et al., 2008) (Melazzi et al., 2012).

Pour résumer, une comparaison entre le SDN et les architectures de réseaux classiques (décentralisées) est faite dans le tableau 1.1. Comme déjà développée, la différence principale entre une architecture SDN et une classique réside dans la séparation des plans de données et de contrôle. Cela permet une configuration plus simple en SDN grâce à la vision globale du contrôleur contrairement à une architecture classique qui nécessite la configuration manuelle de chaque commutateur. Par ailleurs, un contrôle centralisé permet la prise de décision en étant omniscient sur le réseau et ainsi une modification dynamique des décisions alors que dans un réseau classique les décisions sont prises localement. Enfin, d'un point de vue de l'innovation, la centralisation facilite l'implémentation des propositions.

1.2.5 SDN pour la sécurité réseau

La centralisation du contrôle dans un logiciel installé sur un système d'exploitation permet de mobiliser plus de ressource pour le contrôle et donc de faire tourner des algorithmes types Machine Learning (ML). Ces algorithmes peuvent apporter de l'intelligence au contrôleur SDN et répondre à des problématiques telles que la classification du trafic dans un contexte de sécurité ou, plus généralement, pour des algorithmes de contrôle de la Qualité de Service ou Qualité d'Expérience, comme présenté dans (Xie et al., 2018). (Boutaba et al., 2018). Ici, nous allons nous concentrer sur les applications liées à la sécurité. En effet, parmi les nombreux avantages offerts par SDN, il y a la gestion de la sécurité du réseau (Shin et al., 2016). En effet, de par la vue globale offerte par

le contrôleur, il est plus facile de mettre en place des techniques de détection d'intrusion (IDS : intrusion detection system) comme représentée sur la Fig. 1.8. Les techniques classiquement appliquées sont des techniques de détection d'intrusion par anomalies s'appuyant, principalement, sur des techniques de Machine Learning. Nous allons présenter quelques techniques appliquées.

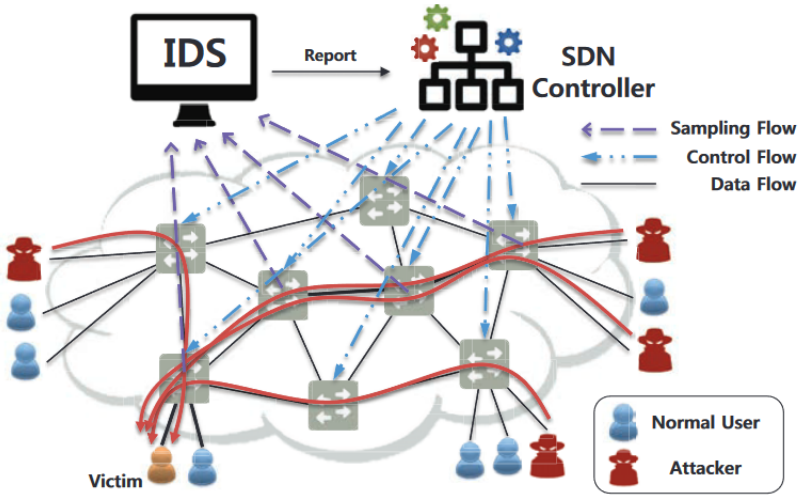


FIGURE 1.8 – IDS classique dans SDN

Le SDN apporte des avantages uniques aux déploiements de solutions de sécurité réseau basées sur le ML comme indiqué dans (Abubakar & Pranggono, 2017), (Herrera & Camargo, 2019) et (Xie et al., 2018). La centralisation du contrôle permet d'avoir une vue globale du réseau et facilite la surveillance ainsi que la collecte des données du réseau. La centralisation du contrôle en SDN permet de réagir immédiatement aux attaques lorsqu'elles sont détectées. Les travaux de (Hurley, Perdomo, & Perez-Pons, 2016) ont proposé d'utiliser le modèle Hidden Markov Model (HMM) pour analyser la structure du trafic à travers les caractéristiques suivantes : longueur du paquet, port source, port destination, IP source et IP destination. De même, (W. Wang, Ke, & Wang, 2018) ont proposé un schéma de détection HMM-R pour détecter les attaques par déni de service distribué (DDoS) en utilisant l'IP source et l'IP destination afin de calculer leur entropie de Rényi (une fonction mathématique généralisée d'entropie qui inclut l'entropie de Shannon et la mini-entropie comme cas particuliers). Un réseau neuronal récurrent d'unités gérées (GRU-RNN) est proposé dans (Tang, Mhamdi, McLernon, Zaidi, & Ghogho, 2018) pour la détection des anomalies. Le GRU-RNN est une technique basée sur le réseau de neurones récurrents (RNN) classique qui peut représenter la relation entre les événements actuels et précédents afin d'améliorer le taux de détection des anomalies. Les caractéristiques considérées sont réparties entre la performance sur le réseau (bande passante et durée) et la structure des paquets (type de protocole, IP source et IP destination). De même (Niyaz, Sun, & Javaid, 2016) a utilisé ce formalisme pour détecter les attaques DDoS dans un réseau SDN. Ils ont utilisé un large ensemble de caractéristiques dérivées des en-têtes de trafic réseau.

D'un point de vue général, toutes les propositions sont similaires dans le sens où elles surveillent le réseau en utilisant la centralisation offerte par l'architecture SDN (comme présenté dans la Fig. 1.8) et recherchent une anomalie en utilisant des prédictions calculées à l'aide d'un algorithme ML. La principale différence réside dans les caractéristiques sélectionnées (qui dépendent principalement de l'attaque surveillée), mais ces dernières peuvent être résumées en

deux catégories : l'observation de la performance (telle que le nombre d'octets transmis) ou de la structure (c'est-à-dire le contenu tel que la source IP, la destination) des paquets. Sur cette base, des algorithmes bien connus sont appliqués (SVM, RNN, HMM, NB, DT ...) afin de distinguer les flux légitimes des flux malveillants.

1.3 Risques liés au contrôle dans une architecture SDN

Néanmoins, le contrôle centralisé, si efficient soit-il, introduit de nouveaux enjeux notamment en termes de sûreté et de sécurité (I. Ahmad et al., 2015). Les spécificités de cette architecture introduisent de nouvelles failles et de nouvelles zones d'attaque, ce qui ouvre les portes à de nouvelles menaces qui étaient plus difficiles à exploiter (Kreutz et al., 2013) (Benton, Camp, & Small, 2013) (Shu et al., 2016) (Chica et al., 2020). Dans un premier temps, nous verrons les différents vecteurs d'attaques sur une architecture SDN et ensuite nous allons développer les attaques qui concernent le cœur de cette thèse : le contrôle.

1.3.1 Menaces générales liées à une architecture SDN

L'architecture SDN apporte des spécificités qui peuvent être utilisées par des attaquants afin de détériorer le service (Hogg, 2014). Par exemple, la centralisation de "l'intelligence du réseau" dans une seule entité a pour conséquence que toute personne, ayant accès aux serveurs qui hébergent le logiciel de contrôle, peut potentiellement contrôler l'ensemble du réseau. Les travaux de (Kreutz et al., 2013) ont identifié sept vecteurs d'attaques principaux comme représentés en Fig. 1.9. De manière générale, il y a une menace selon l'interface mise en place (Shu et al., 2016). Développons chacune des menaces représentées en Fig. 1.9.

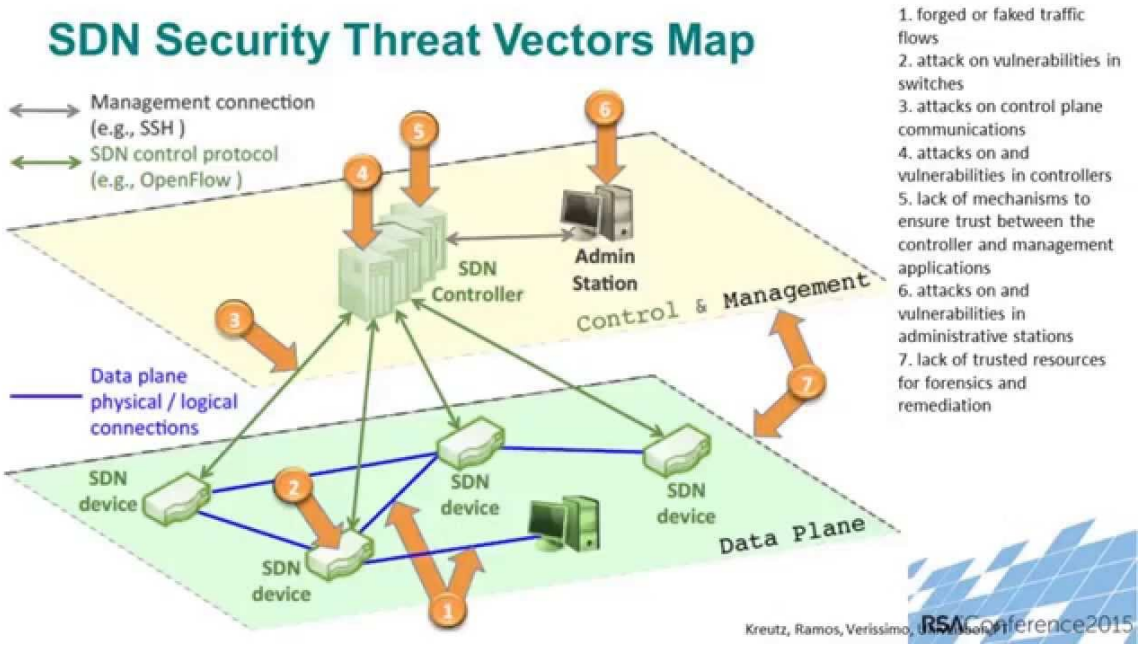


FIGURE 1.9 – Représentation des menaces principales dans une architecture SDN (Kreutz et al., 2013).

1. Les flux au niveau de la couche infrastructure.
   Des flux forgés par un attaquant au niveau des équipements comme l'usurpation d'adresse IP d'un utilisateur.
2. Les équipements de la couche infrastructure.
   Profiter de la vulnérabilité d'un switch pour ralentir ou "perdre" des paquets ou dévier le trafic du réseau (par exemple, à des fins de vol de données). Il est également possible d'injecter du trafic ou de fausses demandes pour surcharger les équipements voisins.
3. L'interface sud.
   Des attaques sur les communications du plan de contrôle, qui peuvent être utilisées pour générer des attaques de type déni de service ou pour le vol de données. Par ailleurs, les paquets peuvent être interceptés et modifiés pour falsifier les données.
4. Le contrôleur.
   Les attaques et les vulnérabilités liées aux contrôleurs constituent probablement les menaces les plus graves pour les réseaux SDN. Un contrôleur défectueux ou malveillant pourrait compromettre un réseau entier, car chaque erreur sera répercutée sur l'ensemble du réseau. À partir du contrôleur, un attaquant peut voler des données, mettre en place un contrôle malveillant, stopper le service sur le réseau ... De même, une application malveillante peut potentiellement faire tout ce qui lui plaît dans le réseau, puisque les contrôleurs ne fournissent que des abstractions qui se traduisent par l'émission de commandes de configuration à l'infrastructure sous-jacente.
5. L'interface est-ouest de communication entre les contrôleurs.
   Cette communication est un enjeu de sécurité puisqu'un contrôleur attaqué peut propager son attaque sur cette interface en transmettant des informations malveillantes sur l'état des commutateurs dont le contrôleur attaqué est responsable. Ces informations erronées auront un impact évident sur les décisions prises par les autres contrôleurs, ce qui pourra endommager le service ou même permettra à l'attaquant de voler des informations sur le réseau.
6. La couche application.
   L'application est construite directement sur le contrôleur et se trouve potentiellement sur le même équipement d'interconnexion physique que le contrôleur. En conséquence, un code malveillant peut être intégré dans le contrôleur via l'application. Par conséquent, l'application est considérée comme le point d'attaque le plus pratique pour s'emparer des contrôleurs.
7. L'absence de ressources de confiance pour l'analyse et le rétablissement après une attaque du réseau.

Ces faiblesses sont utilisées par des attaquants afin de détériorer les performances du réseau ou bien voler des informations. Des travaux répertorient ces attaques comme (Scott-Hayward, O'Callaghan, & Sezer, 2013) ou (Yoon et al., 2017) qui classifient les attaques selon la couche/interface de l'architecture SDN qui est touchée. Pour aller plus loin il y a DELTA, développé par (Lee et al., 2017a) et supporté par l'ONF, qui est un outil répertoriant et permettant de tester un certain nombre d'attaques connues sur les commutateurs SDN.

1.3.2 Focus sur les menaces liées au contrôle de réseau

Plus précisément, ces travaux vont s'intéresser aux menaces qui impactent l'élément clé de cette architecture : le contrôle. Dans une architecture SDN, le contrôleur est la seule entité

décisionnelle sur le réseau et par conséquent, il est clair qu'il peut être hautement ciblé par des attaques. En reprenant la classification de (Kreutz et al., 2013), les menaces sur le contrôle de réseau correspondent aux points 4, 5 et 6 de la Fig. 1.9. Ici, il est important de mentionner que le point 3 ne fait pas partie de nos perspectives puisqu'on considère la sécurité du contrôle uniquement et non la véracité des informations qui remontent au contrôleur. En effet, les attaques peuvent venir de partout et notamment depuis les commutateurs. Des attaquants peuvent forger de faux paquets de type Link Layer Discovery Protocol (LLDP) afin de manipuler les informations liées à la topologie du contrôleur (Hong, Xu, Wang, & Gu, 2015). Dans ce cas le contrôleur prendra de mauvaises décisions d'un point de vue général sur le réseau, mais des décisions correctes du point de vue local de l'activité du contrôle.

Attack		*Vulnerability*		*Examples and Description*
[CP-R] Control Plane Remote attacks				
[CP-R-1]	Denial-of-Service	**[V-3]**	Architectural bottleneck	**i) Packet-In flooding†:** The network hosts participating in an software-defined network may intentionally generate a large number of distinct network flows to exploit the bottleneck that exist in the SDN architecture [63], [10], [54], [33], [59], [60].
		[V-2]	Weak authentication	**ii) Switch table flooding†:** Crafted control messages may be continuously injected to the control plane, and it may eventually fill up the switch table maintained on an SDN controller [12].
				iii) Switch identification spoofing†: Weak switch authentication mechanism may be exploited to remotely drop the legitimate switch connections [11].
		[V-6]	Improper exception handling	**iv) Malformed control message injection:** Malformed control messages may be injected to the control plane to drop the legitimate switch connections.
		[V-8]	Dependence on external variable	**v) System time manipulation:** Arbitrary modification of system time may affect the behavior of SDN controllers.
[CP-R-2]	Network-view manipulation	**[V-2]**	Weak authentication	**i) Host location hijacking†:** Weak host authentication mechanism may be exploited to manipulate the host information [22].
				ii) Link fabrication†: Weak link discovery mechanism may be abused to manipulate the link information [22].
[CP-L] Control Plane Local attacks				
[CP-L-1]	Arbitrary system termination †	**[V-1]** **[V-4]**	Lack of authorization Monolithic controller design	An SDN application may execute a system exit command to terminate the controller instance [53].
[CP-L-2]	System resource exhaustion †	**[V-4]** **[V-5]**	Monolithic controller design Lack of resource management	Poorly designed or malicious SDN components(or applications) may use up the system resources, and ultimately affect the network availability [53].
[CP-L-3]	Network service neutralization	**[V-1]**	Lack of authorization	**i) Control message delivery obstruction I:** A control message subscription list may be manipulated to obstruct arbitrary SDN applications from receiving control messages.
		[V-7]	Naïve service chaining mechanism	**ii) Control message delivery obstruction II:** An SDN application may participate in a service chain and drop control messages before the other applications awaiting for them.
				iii) Service chain jamming: An SDN application may participate in a service chain and freeze (or hold) the sequential execution of services.
[CP-L-4]	Unauthorized application management	**[V-1]**	Lack of authorization	Unauthorized SDN controller components(or applications) may arbitrarily manipulate the state of the target application.
[CP-L-5]	Unauthorized network control	**[V-1]**	Lack of authorization	**i) Flow-rule modification:** An SDN application may issue a flow rule to overwrite the existing rule in the flow table of a switch to cause unexpected network behavior.
				ii) Flow table flushing: An SDN application may flush the flow table entries of a switch to disallow all the communication.
[CP-L-6]	Unauthorized network-view manipulation†	**[V-1]**	Lack of authorization	Unauthorized SDN controller components(or applications) may arbitrarily manipulate the global network view maintained within the control plane [53].

FIGURE 1.10 – Taxonomie non exhaustive des attaques visant la couche contrôle d'une architecture SDN (Yoon et al., 2017)

Les attaques impactant le contrôle ont été largement étudiées dans la littérature puisque leurs conséquences sont à haut risque (Liyanage et al., 2017) (Yoon et al., 2017). Une taxonomie non exhaustive de ces attaques est donnée en Fig. 1.10. Un attaquant tentera de cibler le contrôleur SDN à plusieurs fins. Par exemple, du fait du manque d'authentification matérielle des interfaces nord des différents contrôleurs, un attaquant pourra, directement au niveau du contrôleur, instancier et injecter de nouveaux flux, nouveaux flux malveillants qui échapperaient alors aux différentes politiques de sécurité mises en œuvre. De ce fait, un attaquant peut réussir à mettre

en place des flux malveillants, ce qui laisse la possibilité à l'attaquant de faire circuler les flux à travers les commutateurs à sa guise. En conséquence de cela, l'attaquant pourra ainsi contourner le système et notamment certaines politiques de sécurité.

Par ailleurs, un attaquant pourrait essayer de mettre en place une attaque de type déni de service en visant le contrôleur ou utiliser une autre méthode pour provoquer une oblitération jusqu'à la défaillance du contrôleur. L'attaquant peut essayer de tenter une forme d'attaque de consommation de ressources sur le contrôleur pour le retarder et faire en sorte qu'il réponde extrêmement lentement aux requêtes et la réponse comprend déjà l'envoi des commandes. Cette latence aura un impact sur le service rendu puisque tant que les commutateurs n'ont pas de commandes, ils bloquent les paquets. L'oblitération du simple contrôleur provoque ainsi un ralentissement général sur l'ensemble du réseau. Et pire, une défaillance du contrôleur provoque une immobilisation du réseau.

Enfin, les contrôleurs SDN fonctionnent comme tout logiciel, sur un système d'exploitation (généralement de la distribution Linux). En conséquence, les vulnérabilités de ce système d'exploitation deviennent des vulnérabilités pour le contrôleur. En effet, un attaquant peut utiliser certaines vulnérabilités afin d'ouvrir une session sur la machine et donc avoir un accès direct sur le contrôleur. Dans ce cas, l'attaquant peut directement stopper le contrôleur afin d'immobiliser le réseau, intercepter certains paquets pour éviter leurs traitements ou bien forger ses propres paquets afin de mettre en place le contrôle qu'il souhaite.

À ce stade, nous notons la grande variété des attaques du contrôle de réseau, variété qui nécessiterait d'identifier plusieurs systèmes de détection afin de réponse à chacune d'entre elles et ainsi protéger efficacement le réseau. Dans ce qui suit, nous allons développer des exemples d'attaques sur le contrôle de réseau.

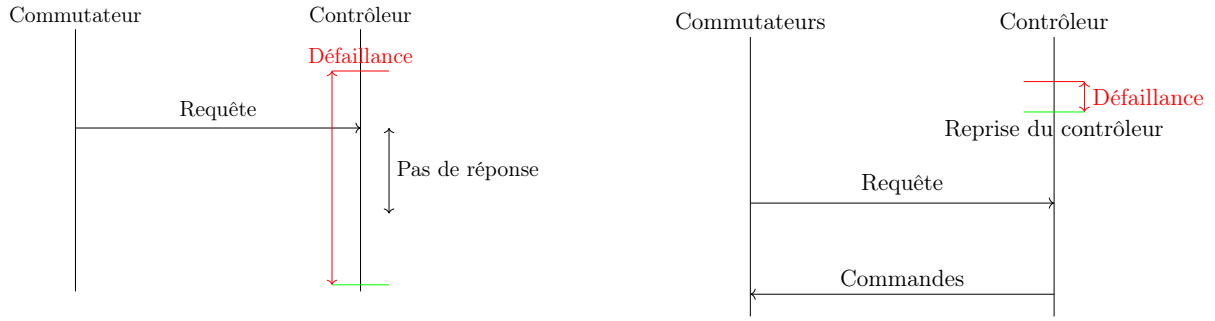
1.3.3 Quelques exemples d'attaques du contrôle d'une architecture SDN

Cette section a pour but de présenter quelques d'attaques et leurs incidences de menaces sur le contrôle de réseau. Tout d'abord nous présenterons le risque de défaillance du contrôleur puis le cas d'une attaque de type déni de service et enfin nous reprendrons quatre attaques ciblant le contrôleur présentées dans (Lee et al., 2017a).

1.3.3.1 Défaillance

L'arrêt du contrôleur, quelle soit due à une attaque ou une défaillance, est la crainte numéro un pour un réseau dans une architecture SDN (Vizarreta, Heegaard, Helvik, Kellerer, & Machuca, 2017) (Pashkov, Shalimov, & Smeliansky, 2014). En effet, en cas de défaillance du contrôleur, tout le réseau est en panne de contrôle. Aucune décision ne peut être prise sur le réseau contrairement à une architecture classique avec un contrôle décentralisé.

Le diagramme de séquence d'une requête d'un commutateur en cas de défaillance du contrôle est donné en Fig. 1.11. Les conséquences décrites en Fig. 1.11a sont claires : les commutateurs auront toujours des requêtes pour le contrôleur qui ne répondra pas. Les paquets au niveau des commutateurs resteront donc bloqués. Cependant, les règles précédemment installées par le contrôleur sont toujours valides. Il est important de mentionner que s'il y a un dysfonctionnement, mais que les commutateurs n'ont pas de requête pendant cette période alors ce n'est pas un problème d'un point de vue du contrôle. Comme représenté en Fig. 1.11b on peut voir que l'état du contrôleur évolue, mais l'activité de la commande n'est pas impacté.



(a) Défaillance du contrôleur causant l'absence de réponse à une requête

(b) Défaillance du contrôleur sans conséquence

FIGURE 1.11 – Diagramme de séquence en cas de défaillance

1.3.3.2 Déni de Service

Le déni de service est une attaque classique dans les réseaux. L'objectif est de surcharger un composant de sorte à le rendre hors service. Dans le cadre d'une architecture SDN il y a plusieurs façons de mettre en place ce type d'attaque (Scott-Hayward et al., 2013) (Lee et al., 2017a). Une classique est de surcharger la table des flux des commutateurs pour qu'il ne puisse plus traiter les nouveaux flux. Ici, on va s'intéresser à celles visant le contrôleur. La conséquence de ce type d'attaque, visant à étouffer à terme le contrôleur, va rapidement s'apparenter au cas de défaillance (associant à la fois les deux comportements de la Fig. 1.11 de manière intermittente). À ce stade, nous ne développerons pas davantage ce cas d'attaque et nous allons nous focaliser la compromission du contrôleur.

1.3.3.3 *Control Message Drop Attack*

Maintenant, nous allons nous focaliser sur des attaques spécifiques à SDN et plus particulièrement ici, d'une attaque appelée *Control Message Drop Attack*. Cette attaque est définie dans (Lee et al., 2017a) par :

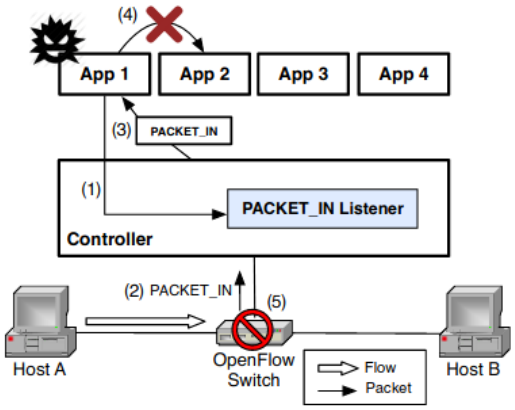
> *Misbehaving or rogue applications can interfere with the order of applications in the list, and cause the application to drop the message (i.e.,Control Message Drop attack).*

La coexistence de plusieurs applications peut introduire des conflits entre elles (P. A. Porras, Cheung, Fong, Skinner, & Yegneswaran, 2015) et ici l'idée de l'attaque est, par manipulation sur l'ordre des applications (Lee, Yoon, & Shin, 2016), de ne pas traiter certaines requêtes. Dans ce cas, certains flux ne seront donc pas servis du tout. Un exemple de diagramme de séquence pour un tel paquet est donné en Fig. 1.12. On y retrouve le processus de l'attaque en Fig. 1.12a divisée en 5 étapes :

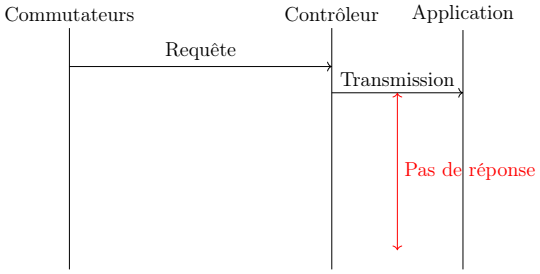
1. Avant la réception d'un paquet, l'application malveillante modifie l'ordre de transmission des requêtes reçues par le contrôleur pour pouvoir intercepter les requêtes.
2. Une requête des commutateurs pour le contrôleur.
3. Transmission de cette requête à l'application malveillante en première suite à la modification de l'ordre.
4. L'application malveillante ne transmet pas la requête pour qu'elle ne soit pas traitée.

5. Le contrôleur ne transmet aucune commande aux commutateurs.

Les conséquences de cette attaque sont représentées en Fig. 1.12b, la requête reste bloquée au niveau des applications et le contrôleur ne transmet pas de commandes.



(a) Processus de l'attaque par une application malveillante (Lee et al., 2016)



(b) Avec attaque

FIGURE 1.12 – Présentation d'une attaque de type *Control Message Drop*.

La cause de cette attaque est la manipulation des applications (ce qui peut même être utilisé pour détourner des applications sécuritaires comme des pare-feux comme développés dans (P. Porras et al., 2012)) tandis que la conséquence est l'absence de contrôle.

1.3.3.4 *Flow Rule Modification Attack*

Considérons une attaque visant à détériorer le service sur le réseau comme le *Flow Rule Modification Attack* définie par (Lee et al., 2017a) de la façon suivante :

A malicious application may also manipulate resident flow rules in the switch that have been installed by other applications. For instance, although a flow rule installed by a firewall application may instruct the switch to drop the flows from the malicious host, a peer application could modify the flow rule to forward corresponding flows from the malicious host (i.e.,Flow Rule Modification attack). Also, by changing the rules, the malicious application can manipulate the flow table in the switch (i.e.,Firmware Misuse and Flow Table Clearance attack).

L'objectif de l'attaquant est de modifier le contenu des tables de flux des commutateurs de sorte à manipuler le contrôle réseau. Par exemple, dans le cadre de routage, un contrôleur malveillant peut modifier les chemins mis en place afin de stopper ou détériorer le service sur le réseau comme représenté en Fig. 1.13. Cela peut être utilisé aussi pour retirer des consignes sécuritaires (type pare-feu).

1.3.3.5 *An Internal Storage Misuse Attack*

Enfin, on développera une dernière attaque appelée *Internal Storage Misuse Attack* définie dans (Lee et al., 2017a) par :

A malicious application may access and alter network topology data within the internal storage of the controller, impacting all peer applications that derive flow control decisions based on this network topology data (i.e., an Internal Storage Misuse attack).

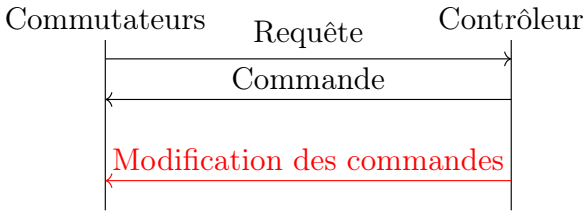
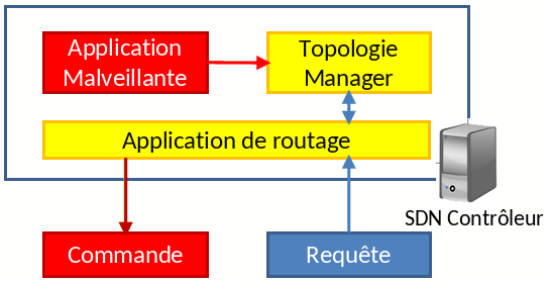
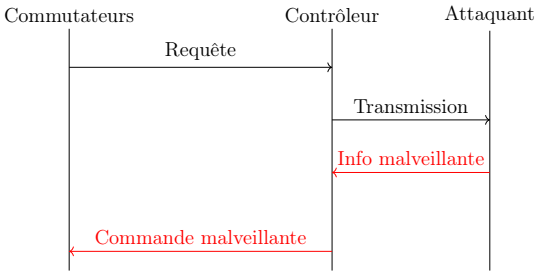


FIGURE 1.13 – Diagramme de séquence du cas d'une attaque Flow Rule Modification. On se place après le régime permanent et l'installation des chemins et autres règles.

L'objectif de l'attaquant est ici de modifier les variables internes du contrôleur afin d'impacter le contrôle. Dans le cas de routage, il est clair que si un attaquant modifie les informations sur la topologie du réseau alors le service sera au mieux pas optimal, au pire pas rendu. Pour ce faire, un attaquant peut lancer une application malveillante qui va corriger ces informations (P. Porras et al., 2012) (P. A. Porras et al., 2015). Le processus est représenté en Fig. 1.14a. Ici, l'application malveillante modifie les informations sur la topologie du contrôleur, ce qui fait que les décisions de routage sont basées sur de mauvaises informations et ne seront donc pas optimales. L'exemple d'un scénario lors d'un tel cas est donné en Fig. 1.14b.



(a) Processus de l'attaque de modifications des variables internes



(b) Diagramme de séquence du cas d'une attaque Flow Rule Modification.

FIGURE 1.14 – Présentation d'une attaque de type *Internal Storage Misuse Attack*

Par ailleurs, la source de l'attaque peut être différente dans une architecture multicontrôleurs puisqu'un contrôleur malveillant peut transmettre des informations erronées aux autres contrôleurs au travers de l'interface est-ouest. Cela aura pour effet de biaiser la vision du contrôleur. Enfin, du fait de l'absence de normalisation de cette interface de communication, la mise en œuvre de ce type d'attaque reste relativement aisée.

1.3.3.6 Synthèse

En synthèse, on peut noter que la liste non exhaustive d'attaques sur le contrôle présentées a des conséquences similaires que nous allons diviser en trois catégories :

— Absence de message du contrôleur (les causes peuvent être la défaillance du contrôleur, une attaque de type Control Message Drop ...)
— Étouffement du contrôleur : Augmentation de la latence de transmission des messages du contrôleur (cette oblitération est généralement due à une attaque de type Déni de Service sur le contrôleur)

— Compromission du contrôleur : mise en place d'un contrôle visant à dégrader le service sur le réseau (ce type d'attaque peut être causé par une attaque de type Internal Storage Misuse ou Flow Mod Modification ...)

1.4 État de l'art des propositions liées à la sécurité et sûreté de la couche contrôle

L'objectif de cette section est de présenter l'état de l'art vis-à-vis des solutions proposées dans la littérature pour résoudre les menaces présentées précédemment. Pour rappel, on propose de catégoriser les menaces et on a obtenu les trois types de menaces : sûreté/défaillance, étouffement et compromission du contrôleur.

1.4.1 Renforcement du contrôleur

Un contrôleur est essentiellement un logiciel. En tant que telles, les techniques d'optimisation logicielle classiques, telles que le parallélisme et le batching, peuvent être utilisées pour améliorer les performances du contrôleur en matière de traitement des demandes, ce qui est utilisé dans Maestro (Cai, Cox, & Ng, 2010) par exemple. Cela permet d'augmenter les performances du contrôleur et d'augmenter sa robustesse contre les attaques de type déni de service. De plus, le contrôleur est logiciel sur un système d'exploitation et fait donc face aux limites et vulnérabilités de ce système d'exploitation. Par conséquent, une proposition de renforcement du contrôleur consiste à renforcer le système d'exploitation sur lequel le contrôleur est installé. Une autre solution, comme proposée dans (Gkountis, Taha, Lloret, & Kambourakis, 2017), est de mettre en place un module de sécurité sur le contrôleur dans le but de vérifier en permanence s'il est la cible d'une attaque de type déni de service ou non. D'autres solutions ont été proposées pour détecter de telles attaques comme dans (Swami, Dave, & Ranga, 2019) (Singh & Behal, 2020). Elles peuvent être divisées en deux catégories : la plus répandue, qui vise à développer un contrôleur de défense basé sur une politique auto statistique, comme le renforcement des actions anti-DDoS en temps réel (RADAR) proposé dans (Zheng et al., 2018) ou (Deng, Gao, Lu, Li, & Gao, 2019). Un module d'extension de SDN nommé DosDefender a été introduit. Il s'agit d'une liste incomplète et le lecteur pourra se référer à (Scott-Hayward et al., 2013) (Chica et al., 2020).

Néanmoins, ces solutions consistent à renforcer le contrôleur pour le prémunir de certaines attaques. Ainsi, le contrôleur est toujours sensible aux problèmes de sûreté liés à la défaillance et, même si c'est dans une moindre mesure, aux attaques de type déni de service. C'est la raison pour laquelle on va se concentrer sur des architectures de contrôle de type multicontrôleurs.

1.4.2 Architecture multicontrôleurs

En effet, l'architecture classique correspond à un unique contrôleur responsable du réseau. Un plan de contrôle physiquement centralisé, composé d'un seul contrôleur pour l'ensemble du réseau, est un choix de conception théoriquement parfait en termes de simplicité. Cependant, un système à contrôleur unique peut ne pas suivre la croissance du réseau. Il est susceptible d'être débordé alors qu'il traite un nombre croissant de demandes et qu'il s'efforce à obtenir les mêmes garanties de performance. De toute évidence, un contrôleur SDN centralisé ne répond pas aux différentes exigences des déploiements de réseaux à grande échelle dans le monde réel. Les centres de données et les réseaux de fournisseurs de services sont des exemples typiques de ces réseaux à grande échelle qui présentent des exigences différentes en termes d'évolutivité et de fiabilité. En effet, en cas de défaillance du contrôleur, les équipements ne recevront plus aucune

commande, ce qui empêche le réseau de pouvoir traiter les nouvelles demandes. Par conséquent, il est nécessaire de proposer une autre conception de la couche contrôle (Hu, Guo, Yi, Baker, & Lan, 2018) (Blial et al., 2016).

Les architectures de plan de contrôle physiquement distribuées on fait l'objet d'une attention accrue de la part des chercheurs ces dernières années, car elles sont apparues comme une solution potentielle pour atténuer les problèmes causés par les architectures SDN centralisées. Il s'agit d'architectures multicontrôleurs qui ont été introduites et proposées (Hu et al., 2018) (Blial et al., 2016). L'objectif est de répartir le contrôle entre plusieurs contrôleurs en proposant une architecture distribuée (que ça soit physiquement ou logiquement). Il y a différentes façons de construire cette architecture : conception horizontale ou hiérarchique du contrôle. Quelle que soit la conception, chaque contrôleur a une vue du domaine auquel il est connecté, et pour assurer une bonne communication des informations liées au réseau, il est nécessaire de mettre en place une interface de communication entre les contrôleurs. Cette interface est classiquement appelée *interface est-ouest*. La conception horizontale, comme initiée dans (Tootoonchian & Ganjali, 2010) et (Koponen et al., 2010) et représentée en Fig. 1.15, étend les capacités du plan de contrôle, mais elle nécessite également une gestion compliquée du contrôleur et des frais généraux de contrôle supplémentaire. Par exemple, les contrôleurs doivent communiquer fréquemment entre eux pour garantir une vue cohérente du réseau. L'organisation des contrôleurs dans un tel style plat présente plusieurs avantages, notamment la réduction de la latence de contrôle et l'amélioration de la résilience (Bannour et al., 2017). La conception hiérarchique est proposée pour résoudre ces problèmes. Dans cette dernière, il y a deux niveaux de contrôleurs : ceux de domaine, qui sont directement connectés aux équipements et exécutent les ordres de la couche supérieure. Enfin, le contrôleur racine, gère les contrôleurs de domaine et maintient une vue globale du réseau. En exemple, Kandoo (Hassas Yeganeh & Ganjali, 2012) est une structure de contrôleur hiérarchique typique. On y retrouve le contrôleur racine qui communique avec les contrôleurs de domaine pour obtenir des informations sur le domaine, mais les contrôleurs de domaine ne communiquent pas entre eux.

De manière générale, ces architectures introduisent un certain nombre de challenges :

1. Position des contrôleurs : le *controller placement problem* (CPP) est un problème d'optimisation répondant aux deux questions suivantes : combien de contrôleurs sont nécessaires et comment définir les domaines ? Plusieurs travaux ont été proposés dans la littérature (Heller, Sherwood, & McKeown, 2012) (Ksentini, Bagaa, Taleb, & Balasingham, 2016) (Sallahi & St-Hilaire, 2014).
2. Répartition de la charge entre les contrôleurs : en raison de de la variation du trafic réseau et de la répartition statique entre équipements et contrôleurs, il est probable que certains contrôleurs soient surchargés et les autres sous-chargés. Par conséquent, une distribution déséquilibrée de la charge entre les contrôleurs va sérieusement impacter les performances du réseau et il est donc essentiel d'assurer une répartition de la charge correcte. Des méthodes variées ont été introduites pour répondre à ce problème (Yonghong et al., 2014) (Selvi, Gür, & Alagöz, 2016) (Cello et al., 2017).
3. Cohérence dans le contrôle : les décisions prises doivent être basées sur les informations cohérentes et en temps réel du réseau. Cependant, pendant la transmission des données entre les contrôleurs (par l'interface est-ouest), la désynchronisation et les conflits stratégiques simultanés des contrôleurs peuvent conduire à l'incohérence du contrôle. En effet, les décisions d'un contrôleur sont basées sur la vue globale du réseau qui provient des informations qu'il a de son sous-domaine directement connecté ainsi que des informations des autres contrôleurs sur leurs parties. Dans le cas de l'équilibrage de charge, si un contrôleur prend

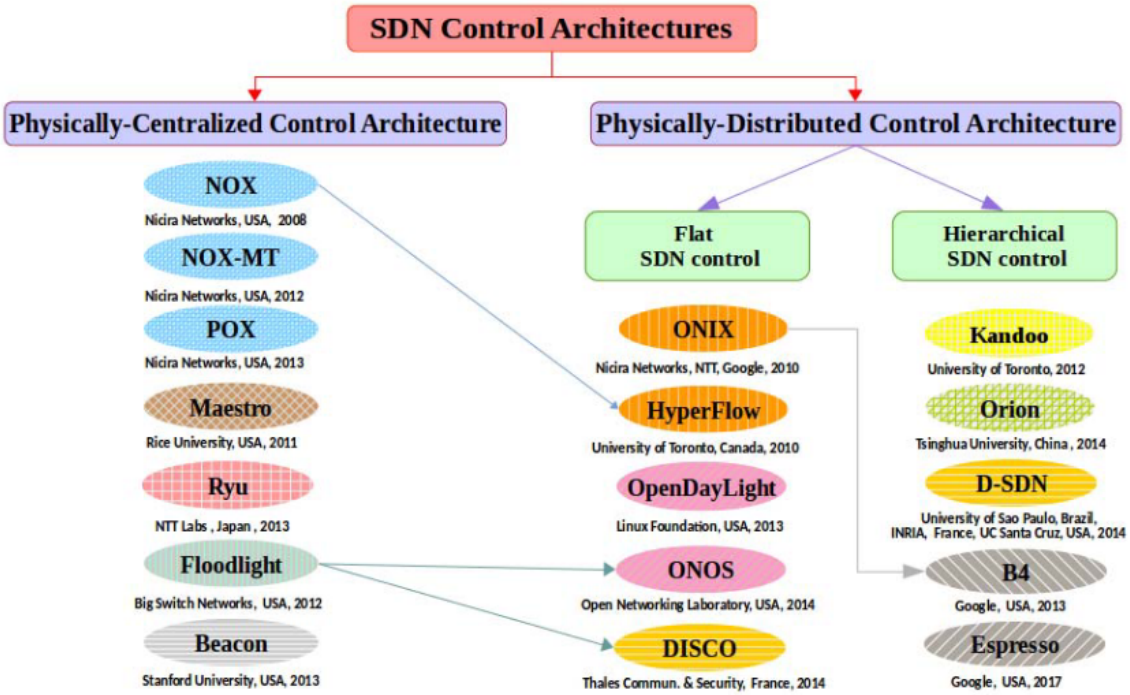


FIGURE 1.15 – Classification physique des architectures du plan de contrôle SDN (Bannour et al., 2017)

une décision avant qu'il ait les dernières informations des autres contrôleurs (par exemple l'arrivée d'une grosse demande) alors ses décisions ne seront pas optimales et pourront impacter cette nouvelle demande (Levin, Wundsam, Heller, Handigol, & Feldmann, 2012) (Guo et al., 2014) (Zhou, Jin, Croft, Caesar, & Godfrey, 2015).

4. Fiabilité du contrôle : les liens de connexion entre les équipements et les contrôleurs ont une capacité limitée. Si ces liens cassent, alors la communication entre les deux couches est interrompue. De plus, bien qu'une architecture multicontrôleurs permet d'offrir une redondance naturelle, l'architecture reste sensible au cas de défaillance ou d'attaque des contrôleurs. L'état de l'art lié à cet enjeu sera détaillé ci-dessous.

En fonction de ces différents choix architecturaux, plusieurs contrôleurs ont été introduits (Bannour et al., 2017). La différence entre chacun des contrôleurs correspond au type de couche contrôle choisi : physiquement centralisée ou distribuée et dans ce dernier cas : logiquement centralisée ou distribuée. À noter que chacun des contrôleurs a des spécificités de programmation qui ont un impact sur ses performances comme étudiées dans (Mostafavi, Hakami, & Paydar, 2020) ou (Mamushiane, Lysko, & Dlamini, 2018) qui proposent une comparaison des performances des contrôleurs en évaluant leurs latences ainsi que leurs débits sur différentes topologies.

1.4.3 Solutions basées sur des architectures multicontrôleurs

Ainsi dans cette thèse, nous allons nous positionner spécifiquement par rapport aux architectures de contrôle qui offre une redondance. C'est le cas des architectures multicontrôleurs, mais également des architectures à un contrôleur et un second composant servant à la sûreté ou la sécurité. Une synthèse des approches est donnée dans le tableau 1.2.

	Menace sur le contrôle				
Travaux	Sûreté	DDOS	Contrôleur malveillant	Rôle du Deuxième Contrôleur	Est-Ouest Communication ?
(Fonseca et al., 2012)	√	√		Détection	Oui
(Das et al., 2020)	√			Détection	Oui
(Y. Wang, Hu, Tang, Xie, & Lu, 2019)		√		Détection	Oui
(Macedo, de Castro, Santos, Ghamri-Doudane, & Nogueira, 2016)	√	√		Détection	Oui
(Rathore, Wook Kwon, & Park, 2019)	√	√		Détection	Oui
(Qi et al., 2016)			√	Validation	Oui
(X. Liu, Xue, Feng, & Dai, 2011)			√	Validation	Oui
(Derhab et al., 2021)	√		√	Validation	Oui
(Yang et al., 2020)	√		√	Validation	Oui
(Lam et al., 2015)			√	Validation	Oui
(Shang et al., 2018)	√		√	Détection	Oui
(Hyder & Ismail, 2021)	√		√	Détection	Oui
Nos travaux	√	√	√	Détection	Non

TABLE 1.2 – Classification des méthodes proposées dans la littérature pour assurer la sécurité et sûreté de la couche contrôle

1.4.3.1 Défaillance des contrôleurs

Le cœur d'un réseau SDN est le plan de contrôle. Toutes les décisions sont gérées par le contrôleur. Ainsi, la défaillance du plan de contrôle a un impact sur l'ensemble du réseau.

La solution classique proposée dans la littérature afin de protéger le réseau d'une défaillance du contrôleur est l'utilisation d'architecture multicontrôleurs comme proposée dans (Fonseca et al., 2012) et (Das et al., 2020). La multiplication des contrôleurs permet de proposer une redondance du contrôleur (active ou passive). Par exemple dans (Fonseca et al., 2012), l'architecture est basée sur l'ajout d'une redondance passive où le second contrôleur doit détecter la défaillance et auquel cas, il doit prendre le relais. Pour détecter une défaillance, lors d'une phase d'inactivité, les commutateurs envoient un message au contrôleur pour vérifier qu'il répond dans les temps. Sinon, le commutateur devra demander au contrôleur de secours. Autre exemple, les travaux de (Das et al., 2020) se fondent sur le principe de redondance active. Le plan de contrôle est un plan distribué qui utilise un mécanisme de synchronisation pour mettre à jour périodiquement l'état de chaque contrôleur au sein de l'architecture de contrôle. En cas de défaillance de l'un d'eux, un autre contrôleur de l'architecture est sélectionné pour prendre le relais.

Cependant, l'implémentation d'une architecture multicontrôleurs implique de nouveaux challenges comme présentés dans (Hu et al., 2018), (Blial et al., 2016) et discuté plus haut.

1.4.3.2 Déni de Service sur les contrôleurs

De même, il est possible d'affecter les performances du contrôleur par un déni de service (DoS) ou un déni de service distribué (DDoS). La motivation d'un DoS ou DDoS est d'inonder les ressources d'un utilisateur prévu pour entraver ou arrêter le service. Étant donné que les commutateurs SDN n'ont aucun contrôle sur les paquets entrants et ne passent pas de temps à les traiter alors l'entité ciblée par ces attaques est le contrôleur (Swami et al., 2019).

À l'aide d'une architecture de contrôle distribuée, un schéma de sauvegarde (Y. Wang et al., 2019) a été proposé pour la protection du plan de contrôle contre les attaques DDoS. Ils ont mis en œuvre une procédure de défense en deux étapes : détection des anomalies de trafic au niveau du plan de données et défense dynamique du contrôleur dans le plan de contrôle. En ce qui concerne la détection ils proposent trois fonctionnalités dans ce module : l'extraction de caractéristiques, la recherche d'anomalies dans ces caractéristiques et l'exécution des résultats. Les caractéristiques considérées ici sont des statistiques sur les flux de la couche infrastructure. Ensuite, la recherche

d'anomalie consiste à déterminer un score de vraisemblance à ces caractéristiques par rapport à un modèle nominal formalisé par un réseau de neurones. Enfin, la défense comprend une nouvelle répartition de la charge entre les contrôleurs et la mise en place d'un contrôle d'accès pour atténuer les attaques DDoS. De même, (Fonseca et al., 2012) a introduit un mécanisme de récupération appelé *CPRevoery* qui est un mécanisme de secours primaire offrant une résilience en cas d'attaques DDoS contre le contrôleur. PATMOS, introduit dans (Macedo et al., 2016), présente un ensemble de procédures pour atténuer les effets des attaques DDoS sur les contrôleurs SDN. Une procédure est divisée en trois phases : identification des contrôleurs surchargés par une attaque, sélection d'un contrôleur maître qui coordonnera le processus de regroupement, et recherche d'une nouvelle configuration. Par ailleurs, une architecture multicontrôleurs basée sur une technologie blockchain a été proposée dans la littérature pour se prémunir contre un point de défaillance unique et les menaces de déni de service, comme dans (Rathore et al., 2019). Ils ont proposé une analyse du trafic pour reconnaître les schémas qui correspondent à un DDoS ou à un flooding TCP. De même, (Sanyal, Barai, & Goplani, 2021) a sécurisé la couche contrôle contre les attaques de type DoS en mettant en place une blockchain entre les contrôleurs.

Cependant, une attaque DDoS peut toujours se propager à travers l'interface entre les contrôleurs et ces méthodes ne tiennent pas compte de la possibilité d'un contrôleur malveillant dans l'architecture qui pourrait transmettre des informations erronées. Ces méthodes sont ainsi sensibles aux menaces liées à l'interface de communication entre les contrôleurs.

1.4.3.3 Attaque sur les contrôleurs

Le réseau est sensible aux perturbations qui affectent le contrôleur, notamment dans le cas d'une attaque qui peut conduire à un contrôleur malveillant. Une telle attaque permet à l'attaquant d'avoir accès à l'ensemble du réseau et de redéfinir la politique de routage précédemment configurée. Face à ces défis de sécurité, différentes solutions ont été proposées dans la littérature pour y faire face. Une architecture à contrôleurs multiples a été proposée dans (Qi et al., 2016) pour prévenir les comportements non sécurisés. L'objectif est de prévenir une attaque en permettant à tous les contrôleurs de s'observer mutuellement. En particulier, il s'agit de déterminer si les règles issues d'un contrôleur sont valides. Dans cet objectif, il y a un vote entre tous les autres contrôleurs, ce qui limite l'influence d'une attaque de contrôleur. Chaque contrôleur participe à la détermination du contrôleur infecté en validant sa production. De manière similaire, (X. Liu et al., 2011) a proposé de configurer un filtre qui valide les commandes envoyées par le contrôleur. Un second contrôleur, jouant le rôle de filtre, a été ajouté pour recevoir les commandes du contrôleur en provenance des commutateurs afin de les valider. Récemment, la blockchain a également été considérée comme une option pour sécuriser les architectures SDN, (Alharbi, 2020) et (Nam Nguyen, Anh Tran, Fowler, & Souihi, 2021), et plus particulièrement la couche de contrôle. En particulier, l'interface de communication entre les contrôleurs, comme dans (Derhab et al., 2021) ou (Yang et al., 2020). Les décisions des contrôleurs font l'objet d'un vote entre tous les autres contrôleurs pour assurer la cohérence des décisions.

Cependant, ces observations ont été réalisées grâce à la communication entre les contrôleurs. Cette communication est typique d'une approche multicontrôleurs et est établie par une interface est-ouest. Une telle interface constitue une faiblesse, car un contrôleur malveillant peut y diffuser des informations incorrectes. Pour résoudre ce problème, (Lam et al., 2015) a proposé l'introduction d'un générateur de clés privées dans le plan de contrôle pour chiffrer les communications entre les contrôleurs. L'objectif est de sécuriser le canal de communication utilisé pour l'interface est-ouest. De même, (Shang et al., 2018) a proposé une architecture globale sécurisée et s'est concentré sur le développement d'un mécanisme de communication sécurisé entre les

contrôleurs. Ils ont utilisé une méthode de communication indirecte en utilisant le "flux d'agents inter-domaines" qui constitue un tunnel de communication sécurisé. En outre, une approche multigranulaire des défis de sécurité et de sûreté du contrôleur a été proposée dans (Shang et al., 2018).

Une autre solution proposée dans la littérature est l'utilisation de la défense par cibles mobiles, comme dans (Hyder & Ismail, 2021) avec l'introduction de la notion de contrôleurs fantômes. De plus, en cas de détection d'un trafic malveillant visant les contrôleurs, une partie des contrôleurs fantômes est sélectionnée aléatoirement pour répondre au trafic.

1.4.3.4 Synthèse

En résumé, l'approche multicontrôleurs permet de traiter des menaces liées à la sûreté et la sécurité de la couche contrôle en ajoutant des renforts, qui peuvent avoir un rôle spécifique dans les traitements de sécurité et de sûreté, comme dans les méthodes présentées. De manière générale, l'apport d'un autre contrôleur au côté d'un contrôle principal permet d'attribuer une tâche spécifique liée à la sécurité ou la sûreté. On a synthétisé ces tâches en deux catégories : validation des décisions ou bien détection d'une anomalie. Pour mettre en place ces tâches, les travaux présents dans la littérature se basent sur la communication entre les contrôleurs, ce qui est une menace de sécurité comme on a pu le voir. Ainsi, les méthodes présentées sont sensibles aux problèmes de sécurité liés à l'interface est-ouest. En effet, chacune des propositions de renforcement de l'architecture de contrôle se base sur une communication entre les contrôleurs. Alors, parfois cette communication est indirecte ou renforcée, mais elle existe toujours. Cette interface est une menace de sécurité comme présentée dans (Kreutz et al., 2013) et en Fig. 1.9. Supposons qu'un contrôleur soit attaqué et qu'un attaquant ait réussi à reprendre la main sur lui. Les autres contrôleurs ont besoin de ses informations afin de pouvoir prendre des décisions et, à la suite de l'attaque, les contrôleurs recevront des informations forgées par l'attaquant qui sont malveillantes. Le risque est donc la prise de décisions basées sur des informations falsifiées du sous-domaine du contrôleur attaqué, une corruption lors de prise de décisions liées à la sécurité comme dans une blockchain par exemple puisqu'en ciblant les pairs de la blockchain, les attaquants peuvent facilement prendre le contrôle du processus de consensus et arrêter les opérations de la blockchain (Abou El Houda, Hafid, & Khoukhi, 2020). Cette menace n'est pas évitée en chiffrant la communication entre les contrôleurs ou en passant par un biais puisque la véritable menace réside dans le contenu de l'information transmise même si cela peut compliquer la tâche de l'attaquant. Par conséquent, l'objectif de cette thèse est d'introduire une architecture multicontrôleurs sans interface est-ouest.

L'objet de cette thèse est de détecter les attaques ayant pour conséquence ce type d'anomalies. En effet, nous ne travaillons pas sur l'isolation de l'attaque à cause de la grande diversité des attaques. Il est également important de mentionner le risque des attaques dites "zero-day" profitant des faiblesses non découvertes comme (Al-Rushdan, Shurman, Alnabelsi, & Althebyan, 2019) qui a profité d'une faiblesse d'exploitation dans le code du contrôleur. De ce fait, la détection portera sur les conséquences de l'attaque et non l'isolement des causes. Ces conséquences concernent l'activité du contrôle c'est-à-dire les échanges entre le contrôleur et les commutateurs alors que les causes correspondent à l'état du contrôleur. Ainsi, même si le contrôleur est attaqué, mais que l'activité de la commande ne présente pas d'anomalies alors du point de vue de notre méthode de détection, attendu que le contrôle est correctement assuré sur le réseau, nous ne chercherons pas à relever une alarme (même si cette compromission n'est pas sans futures conséquences).

1.5 Conclusion

Pour conclure ce chapitre, nous avons pu voir l'intérêt d'un contrôle centralisé et particulièrement l'architecture Software-Defined Networking qui sépare le contrôle des équipements d'interconnexion. Cette architecture présente de nombreux bénéfices, mais également de nouveaux enjeux, notamment en termes de sécurité. On a pu voir que la cible privilégiée de cette architecture est le contrôleur de par son rôle central. En plus de ces menaces de sécurité, le réseau est également sujet à des menaces liées à la sûreté du contrôleur. En réponse à ces problématiques, l'architecture multicontrôleurs a été introduite. Diverses solutions ont été proposées dans la littérature afin de renforcer la couche contrôle en utilisant une architecture de type multi-contrôleurs. Néanmoins, nous avons identifié une faiblesse aux solutions proposées : la présence d'une interface de communication entre les contrôleurs (Menace numéro 5 d'une architecture SDN présentée dans (Kreutz et al., 2013) et résumé plus haut). C'est la raison pour laquelle nous allons introduire une architecture multicontrôleurs sans une telle interface et nous allons la présenter dans le chapitre suivant et formaliser la problématique scientifique inhérente à ce type de système.

Chapitre 2

Problème de détection : état de l'art et première proposition

Dans ce chapitre nous allons formaliser le problème de détection d'anomalies de contrôle sans interface est-ouest. Dans un premier temps, l'architecture et les différents éléments, ainsi que leurs rôles, sont introduits. Il y aura un contrôleur en charge du contrôle et un observateur chargé de détecter les anomalies dans le contrôle. Le problème de capture de l'observateur est présenté dans une seconde section. Puis, le principe de la méthode de détection de l'observateur est présenté et enfin un premier observateur sera mis en place.

2.1 L'architecture de contrôle proposée

Cette section développe l'architecture proposée. Comme évoqué, l'originalité de ces travaux est de proposer une architecture sans interface est-ouest, c'est-à-dire sans interface de communication entre les éléments de la couche contrôle. L'architecture proposée est présentée en Fig. 2.1. L'architecture est composée de :

— Un contrôleur $c0$ en charge du contrôle sur le réseau.
— Un observateur O en charge de la détection d'anomalies dans le contrôle

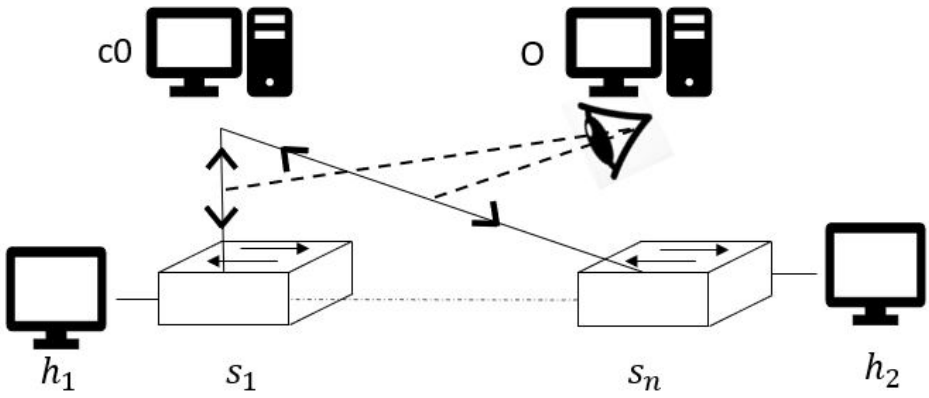


FIGURE 2.1 – Architecture de contrôle proposée

En conséquence de l'absence d'interface est-ouest, la méthode de détection ne sera pas basée sur la surveillance directe des états internes du contrôleur principal puisque nous n'y avons pas

accès, mais sur l'observation de son activité sur le réseau et une certaine connaissance *a priori* de la logique de contrôle. L'observateur doit donc faire face à un problème de capture puisqu'il aura accès uniquement aux messages situés au niveau de l'interface sud et qu'à partir de ces messages il aura pour tâche de reconstruire la solution de contrôle. Cela avec pour objectif de déterminer si cette solution présente des anomalies. En outre, nous considérerons que le contrôleur est chargé avec une application de découverte de la topologie qui fonctionne également sur l'observateur, ce qui permettra au second contrôleur d'avoir accès à $\mathcal{G}$ le graphe de la topologie de l'infrastructure.

Nous allons développer ci-dessous le rôle de chacun des éléments de cette nouvelle couche contrôle.

2.1.1 Le contrôleur

Le premier contrôleur est en charge du contrôle du réseau. Le contrôleur reçoit des requêtes provenant des infrastructures permettant de déterminer les flux à mettre en place sur le réseau. En plus de cela, il reçoit des statistiques qui peuvent permettre de comparer l'état sur le réseau à la consigne dictée par l'application et déterminer quelles commandes mettre en place. Le processus de commande est représenté en Fig. 2.2.

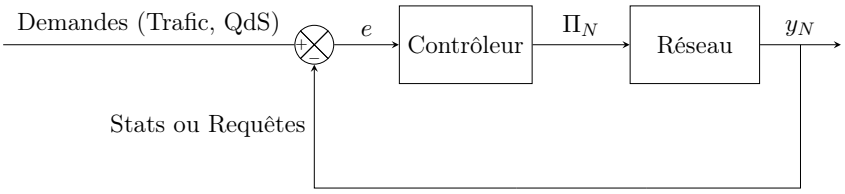


FIGURE 2.2 – Boucle de contrôle

À titre illustratif, nous considérons la fonction d'acheminement des paquets (on notera néanmoins que la même méthodologie s'appliquera pour d'autres fonctions réseau). Étant donné cette application, la tâche du contrôleur va être de mettre en place le plan de données Π_N sur le réseau en fonction des requêtes et/ou statistiques renvoyées par les commutateurs. Selon l'algorithme de commande, les statistiques ne sont pas forcément utilisées. Par exemple, en considérant un algorithme visant à choisir le chemin le plus court, en matière de nombre de saut, en utilisant Dijkstra ou Belleman-Ford alors les statistiques ne sont pas utilisées dans le processus de décision. Dans ce qui suit, nous allons définir ce qu'est un plan de données. De manière générale, un plan de données correspond aux chemins mis en place. Ces chemins s'inscrivent donc dans une topologie formalisée comme suit.

2.1.1.1 Plan de données

La topologie est définie par un graphe dirigé $G = (V, E)$, comprenant un ensemble V de sommets $i \in \{1, 2, \dots |V[= n\}$ ainsi qu'un ensemble $E \in V^2$ d'arêtes (i, j), qui sont des sous-ensembles à deux éléments de V. Pour un graphe simple avec un ensemble de sommets V, la matrice d'adjacence est une matrice A carrée de dimension $|V| \times |V|$ telle que son élément $A_{i,j}$ vaut 1 lorsqu'un lien existe entre deux nœuds i et j et 0 lorsqu'il n'y a pas d'arête. On considérera que les éléments diagonaux de la matrice sont tous nuls.

Le graphe G représente la topologie réseau. Nous introduisons maintenant la partie dynamique, c'est-à-dire les demandes de trafic. Considérons un ensemble $\mathcal{K} = 1, 2, \dots, K$ de couples Origine-Destination. Soit s_k et d_k la source et la destination de la k-ième demande et soit $\lambda_k \in \mathbb{R}^+$

la demande de trafic (égale ici au débit en b/s). La couche contrôle est chargée de déterminer le chemin à mettre en place pour chaque demande. Généralement, plusieurs chemins peuvent être disponibles pour une paire source-destination donnée et c'est donc au contrôleur de choisir ceux qui respectent la stratégie de routage dictée par la couche application. Soit Π, le plan de donnée, une matrice qui rassemblera les décisions de routage pour toutes les demandes. L'ensemble Π est défini de telle sorte que $\Pi_{k,v}$ donne le nœud adjacent au nœud v pour la demande k. La k-ième ligne de Π représente donc le chemin p_k mis en place par le contrôleur pour la demande numéro k. Un chemin p_k peut donc être facilement extrait de Π puisque $p_k = \{s_k, \Pi_{k,s_k}, \Pi_{k,\Pi_{k,s_k}} \ldots, d_k\}$.

Le choix de ces différents chemins se fait selon une politique de routage visant à respecter un certain nombre de contraintes comme la capacité des liens et visant aussi à minimiser/maximiser un certain nombre de métriques en lien avec la Qualité de Service requise sur le réseau (Georges, 2019). D'autant que les spécificités offertes par l'architecture SDN sont propices aux développements de contrôle visant à améliorer la qualité de service sur le réseau en utilisant des algorithmes de type Machine Learning comme l'apprentissage par renforcement (Sellami, Hakiri, Yahia, & Berthou, 2022) (Casas-Velasco et al., 2020). Cependant, le choix de la politique de routage ainsi que sa mise en place ne sont pas l'objet de ces travaux.

Prenons un exemple pour illustrer cette définition. Considérons une topologie simple de 6 infrastructures comme donnée en Fig. 2.3 ainsi que 4 demandes à l'instant t tel que : $\mathcal{K} = \{(1,5), (1,6), (2,6), (3,4)\}$

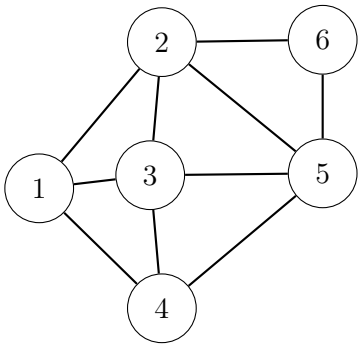


FIGURE 2.3 – Topologie de 6 switchs

Un exemple de plan de données Π_N est donné ci-dessous. Considérons la demande $k_1 = (1,5)$. Ici, le choix du contrôleur est de faire passer le flux par le commutateur 2. Ainsi, $s_{k_1} = 1$, $\Pi_{k_1,s_{k_1}} = 2$, $\Pi_{k_1,\Pi_{k_1,s_{k_1}}} = d_{k_1} = 5$ et donc : $p_{k_1} = \{1,2,5\}$

$$\Pi_N = \begin{pmatrix} 2 & 5 & 0 & 0 & 0 & 0 \\ 2 & 6 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 6 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \end{pmatrix}$$

2.1.1.2 Cas de menace

En cas d'attaque, le contrôle sera biaisé comme symboliquement représenté en Fig. 2.4. En effet, on ne s'intéresse pas aux raisons de l'attaque et on considère uniquement les conséquences. Comme vu dans le chapitre précédent, les conséquences sont de trois types : absence de commande, retard dans les commandes ou modification des commandes. En conséquence, on modélise

l'ensemble des attaques sur le contrôle sous la forme d'un biais de la commande. Ainsi, les commandes envoyées aux infrastructures en cas d'attaque, quelle que soit l'attaque, seront biaisées. L'attaquant peut mettre en place un biais de sorte à changer la stratégie de contrôle par exemple. Au lieu du plan de données nominal Π_N calculé par le contrôleur, ce sera a priori un autre plan Π_A qui sera transmis aux commutateurs. Il n'est pas à exclure que l'attaquant mette en place un plan de données nominal auquel cas il n'y pas d'anomalies vis-à-vis du contrôle. Ce biais peut se traduire de diverses façons comme nous le verrons lorsque nous le formaliserons ci-après.

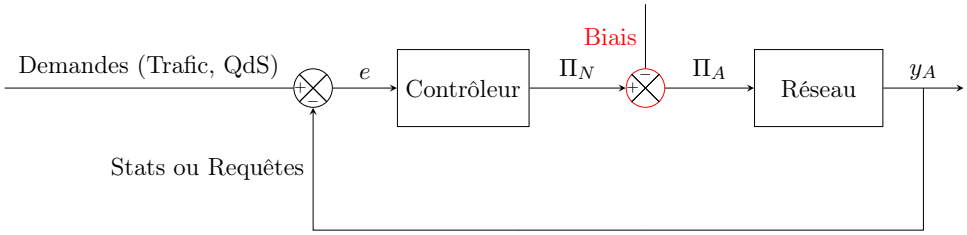


FIGURE 2.4 – Boucle de contrôle biaisé

2.1.2 L'observateur

Nous venons de présenter le contrôleur est responsable du contrôle de réseau. Comme mentionné, on introduit un observateur en charge d'observer l'activité de la commande. Le but est de vérifier qu'il n'y a pas d'anomalies liées à un problème de sûreté ou de sécurité comme représenté sur Fig. 2.5.

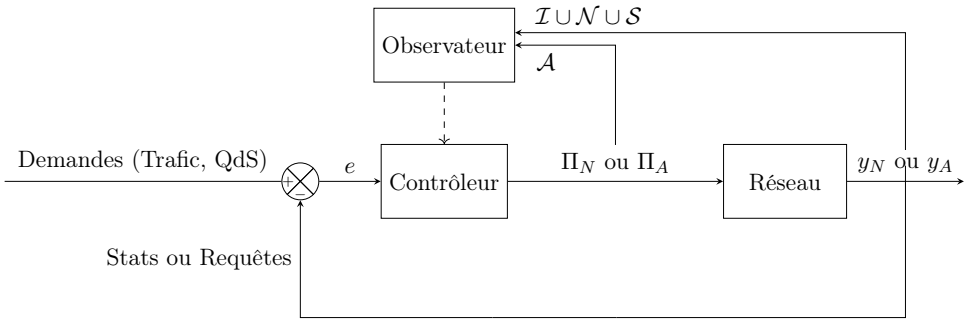


FIGURE 2.5 – Boucle de contrôle avec l'observateur

L'observateur est placé de sorte à avoir accès à l'activité de la commande. Cette activité est composée de l'ensemble des paquets qui constituent le plan de données, c'est-à-dire les commandes de $\mathcal{A}$, ainsi que les requêtes et/ou statistiques provenant des infrastructures à savoir $\mathcal{I} \cup \mathcal{N} \cup \mathcal{S}$. Les ensembles $\mathcal{A}$, $\mathcal{I}$, $\mathcal{N}$ et $\mathcal{S}$ sont formalisés en section. 2.2.1. De manière générale, cela correspond à l'ensemble des paquets passant par l'interface sud.

À partir de cette trace de l'activité de la commande, l'observateur devra vérifier l'absence d'anomalies. Pour cela, notre observateur aura pour tâche de reconstruire le plan de données uniquement à partir des commandes, des requêtes et des statistiques observées et de vérifier, connaissant l'algorithme de contrôle, s'il n'y a pas d'anomalies dans le contrôle et que le plan de données est issu de l'algorithme de contrôle ou bien s'il est biaisé. En effet, ce n'est pas l'état

du contrôleur qui est évalué, mais l'état de l'activité de la commande et donc l'observateur est responsable de la détection de biais dans le contrôle et non de l'évaluation de l'état du contrôleur à tout instant.

Il est important de noter que la reconstruction du plan de données n'est pas toujours possible ou bien trop coûteuse. Ici, on fait l'hypothèse qu'on ait accès à l'ensemble des paquets permettant cette reconstruction.

Dans cet objectif, l'observateur est placé physiquement dans l'ombre (c'est-à-dire qu'il n'envoie aucun paquet sur le réseau et n'interfère donc pas sur le contrôle de réseau) et observe uniquement les messages au niveau de l'interface sud.

2.2 Construction du problème de capture

Nous venons de présenter l'architecture de contrôle proposée. L'observateur a donc pour tâche de détecter toutes anomalies dans le contrôle par unique observation de l'activité de la commande correspondant aux paquets échangés au niveau de l'interface sud. À partir de ces messages, l'observateur devra reconstruire le plan de données afin de vérifier qu'il n'y a pas d'anomalies. Dans un premier temps nous allons formaliser les paquets observés par l'observateur pour reconstruire formellement le plan de données et enfin on développera le principe de détection qui va être mis en place.

2.2.1 Formalisation des primitives du protocole OpenFlow

L'observateur surveille l'interface sud normalisée par un protocole développé par l'ONF *OpenFlow*. En effet, les paquets OpenFlow (version 1.3, (ONF, Juin, 2012)) échangés, entre le contrôleur et les commutateurs, sont représentatifs du comportement réel du contrôleur. L'ensemble de l'activité (c'est-à-dire la trace Σ) du contrôle est défini en conséquence comme l'ensemble des paquets :

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}, \qquad \sigma_i = (t, \text{type}, \ldots)$$

et un paquet, σ_i noté comme un n-tuple, se compose de la date de la capture t, du type OpenFlow et de champs supplémentaires spécifiques (liés au type) comme décrits dans ce qui suit.

Parmi tous les types de paquets OpenFlow, l'activité du contrôle est principalement déterminée en fonction de quatre types : les entrées (requêtes des infrastructures au contrôleur), les commandes (envoi du paquet à partir d'un port spécifié, modification de l'état du commutateur), le statut (modification du statut du port/de la liaison d'un commutateur) et les statistiques (lecture des informations d'état d'un commutateur concernant le flux, la table, la file d'attente, etc.). Développons chacune de ces fonctions.

Lorsqu'un commutateur reçoit pour la première fois un paquet d'un flux donné, il envoie un message `Packet_In` au contrôleur afin de connaître l'action à exécuter pour ce paquet. On note donc $\mathcal{I} \subset \Sigma$ l'ensemble des `Packet_In` tels que :

$$\forall \sigma \in \mathcal{I}, \sigma = (t, \texttt{"Packet\_In"}, \omega, \rho, s, d)$$

où ω correspond à l'identifiant du commutateur, ρ au port d'entrée du commutateur, s et d les adresses IP source et destination des paquets.

Les actions $\mathcal{A}$ (par exemple supprimer le paquet ou le retransmettre sur un port spécifié) peuvent être émises selon deux types de commandes. Tout d'abord, la commande `Packet_Out`

qui n'est appliquée qu'une seule fois par le commutateur qui ne conserve pas l'information et devra redemander au contrôleur si un paquet similaire est reçu. On défini ainsi :

$$\mathcal{I} \in \Sigma \mid \ \forall \sigma \in \mathcal{A}_p, \sigma = (t, \texttt{"Packet\_Out"}, \omega, \rho, s, d)$$

où ω est l'identifiant du commutateur sur lequel l'action doit être appliquée, ρ est l'identifiant du ou des ports sur lesquels le paquet (de s à d) doit être transmis.

La deuxième commande est `Flow_Mod`, qui est retenue temporairement par le commutateur qui ajoute cette commande à sa table de flux et appliquera directement l'action pour tout paquet correspondant. On peut ainsi définir :

$$\forall \sigma \in \mathcal{A}_f, \sigma = (t, \texttt{"Flow\_Mod"}, \omega, \rho, s, d, \delta, \text{type})$$

où ω, ρ, s et d sont identiques à ceux de $\mathcal{A}_p$ et δ est la durée de vie de l'action suivie du type de l'action (c'est-à-dire ajouter, modifier ou supprimer une règle).

Les paquets de statuts consistent en des notifications des commutateurs concernant le statut de leurs ports (message `Port_Status`), des liens ou du commutateur lui-même. Il s'agit d'informations liées à la topologie et donc au graphe $\mathcal{G}$. Nous nous concentrons ici sur le statut des ports, de sorte que nous notons $\mathcal{N}$ l'ensemble de ces messages tel que :

$$\forall \sigma \in \mathcal{N}, \sigma = (t, \texttt{"Port\_Status"}, \omega, \rho, \text{type})$$

où le type correspond à la raison du paquet (`add` pour notifier un nouveau port, `delete` et `modify` un changement du statut du port donné ρ.

Le dernier type de paquets correspond aux statistiques envoyées par les commutateurs (`"MultipartReply"`) en réponse à la demande du contrôleur (par `"MultipartRequest"`). Selon le protocole OpenFlow v1.3 (ONF, Juin, 2012), il existe plusieurs types de statistiques fournies par le commutateur et dans ce travail, nous ne considérerons que les statistiques liées au flux. On définit ainsi :

$$\forall \sigma \in \mathcal{S}, \sigma = (t, \texttt{"MultipartReply"}, \omega, s, d, b)$$

tel que b est le nombre total d'octets, concernant le flux de s à d, retransmis par le commutateur ω.

Au final, cela signifie que la trace Σ (c'est-à-dire l'ensemble des paquets capturés par l'observateur) est constituée de l'ensemble des entrées, actions, statuts et statistiques, de sorte que :

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\} = \mathcal{I} \cup \mathcal{A} \cup \mathcal{N} \cup \mathcal{S}$$

Une attention particulière est portée sur les actions puisqu'elles sont consécutives à la logique du contrôleur. Un contrôleur malveillant ou défectueux entraînera des actions manquantes ou inattendues. Le rôle d'un observateur sera donc de déterminer si les actions capturées sont, compte tenu du contexte (les demandes, la topologie, les entrées, l'état et les statistiques), cohérentes et plausibles avec un contrôle non défectueux et non piraté (et donc si la trace est cohérente ou malveillante).

2.2.2 Reconstruction du plan de données

L'analyse de cette activité dépend de la fonction de contrôle considéré. Dans ces travaux, nous illustrons notre méthodologie sur la base d'une fonction de routage. Cela signifie que l'action consiste en des décisions d'acheminement (vers quel(s) port(s) un commutateur doit-il retransmettre un paquet ?) comme développé plus haut. C'est-à-dire que pour mettre en place un

plan de données Π, le contrôleur utilise les paquets de types `Flow_Mod` afin de mettre en place les règles.

Donc, du point de vue de l'observateur, en considérant une demande de s à d, cela signifie que la route installée est constituée des actions envoyées par le contrôleur et peut être identifiée par la fonction (surjective) suivante :

$$\mu(t, s, d) = \{\sigma \in \mathcal{A} \mid t_\sigma \in [t - \delta, t], s_\sigma = s, d_\sigma = d\}$$

où δ correspond à la durée de vie des actions .

La fonction μ permet de reconstruire une route. Généralisons maintenant au plan de données en considérant la liste des demandes $\mathcal{D}$ pour lesquelles le contrôleur a décidé d'installer des routes à l'instant t telle que :

$$\mathcal{D}(t) = \{(s, d) \mid \exists \sigma \in \mathcal{A}, t_\sigma \in [t - \delta, t], s_\sigma = s, d_\sigma = d\}$$

À noter qu'il est important de prendre en compte le temps de vie de ces actions (δ). En effet, une action mise en place à $t - \delta + \epsilon$ est toujours valable à t.

Cela permet finalement de reconstruire, au niveau de l'observateur, le plan de données $\mathcal{P}$ (c'est-à-dire l'ensemble des routes actives) à un instant donné t comme :

$$\mathcal{P}(t) = \overset{*}{\bigcup_{\forall (s,d) \in \mathcal{D}(t)}} \mu(t, s, d)$$

où $\overset{*}{\bigcup}$ consiste en un opérateur d'union spéciale considérant les différents types d'actions. Si le type du `Flow_Mod` est `add`, l'action (de transfert) est directement ajoutée à l'ensemble ; si le type est `delete`, l'action précédente associée est supprimée de l'ensemble et si le type est `modify`, l'action précédente associée dans l'ensemble est remplacée par la nouvelle règle.

Ainsi, uniquement par observation de l'activité de la commande, l'observateur reconstruit le plan de données puisqu'il correspond aux commandes envoyées par le contrôleur. Par ailleurs, dans ces travaux, on fait l'hypothèse qu'il n'y a pas de problèmes de capture et donc l'observateur a accès à l'ensemble des paquets. Ainsi, le plan de données Π mis en place par le contrôleur est totalement reconstruit par l'observateur et est donc, sans attaque et sans défaillance, similaire à $\mathcal{P}$. Le but de l'observateur consiste ainsi, par définition (sans préjuger de la solution) et à partir de cette reconstruction, à déterminer s'il est représentatif d'un contrôle sans défaillance et sans attaque.

Notons d'ores et déjà que l'hypothèse d'une capture parfaite pose le problème de l'observabilité de réseaux, à savoir la capacité pour l'observateur à détecter les attaques/défaillances en cas d'informations manquantes.

2.2.3 Explicitation de l'impact d'une menace

Comme présenté plus haut, dans le cas d'attaque ou bien de défaillance l'algorithme de contrôle retournera une commande biaisée. Ces menaces peuvent avoir plusieurs sources comme vues dans le chapitre 1. Toutefois, nous ne nous intéresserons pas ici à réparer le contrôleur, donc pas à l'isolement de la faute, mais simplement, à la détection (afin de pouvoir en définitive à rebasculer sur un autre contrôleur). Ainsi on ne s'intéresse pas à isoler la faute et on se concentre donc sur les conséquences. Comme évoqué précédemment, les conséquences sont regroupées selon de trois types :

— Terminaison de la communication entre le contrôleur et les commutateurs (défaillance du contrôleur),
— Étouffement du contrôleur,
— Compromission des décisions prises par le contrôleur.

Formellement, on a modélisé cet impact par un biais, que l'on appellera b_{Cmd}, qui convertit une commande $pout_N \in \mathcal{A}$ en une commande biaisée $pout_A \in \mathcal{A}$ tel que :

$$pout_A = pout_N + b_{Cmd}$$

De sorte que :

$$\forall i \in [1, len(pout)] pout_A[i] = pout_N[i] + b_{Cmd}[i]$$

C'est-à-dire que le biais peut impacter n'importe quel élément du paquet afin de le modifier pour perturber le contrôle. Classiquement, on s'intéressera à deux modifications particulières : celle du temps de transmission t (augmentation de la latence pour cause d'attaque de type déni de service par exemple) et celle du port de transmission ρ (pour modifier la décision du contrôle qu'aurait été celle prise par l'algorithme nominal). De plus, il y a le cas particulier $pout_A = \emptyset$ correspondant à l'absence de commande et donc au cas de terminaison de communication entre le contrôleur et les commutateurs. Ce cas particulier est à mettre en relation avec le cas de défaillance.

L'objet de ces travaux est de détecter ces biais. Dans ce qui suit, nous allons présenter le principe de notre méthode de détection.

2.3 État de l'art des techniques de détection

Nous venons de poser le problème de détection lié à notre architecture. Il est important de rappeler qu'on ne s'intéresse qu'à la détection de la faute et non à son isolement. On cherche seulement à savoir si le contrôleur a un souci, de type sûreté ou sécurité, et non identifier l'attaque. Il existe plusieurs façons de résoudre ce type de problème et nous allons développer ces méthodes pour nous positionner. Ces méthodes peuvent être divisées en deux catégories : celle reposant sur une connaissance experte de l'attaque et celle reposant sur la recherche d'anomalies dans le comportement. Dans les deux cas, l'objectif est de déterminer un modèle du système (fautif ou non) puis de comparer avec les observations du système réel. Le problème majeur repose sur la construction de ce modèle : inclure les comportements fautifs ou non.

2.3.1 Techniques basées sur la connaissance de l'attaque

Tout d'abord, la technique la plus intuitive consiste à construire le modèle fautif du système. Ainsi, l'hypothèse de ces travaux est la connaissance de l'attaque. À partir de cette connaissance, un modèle fautif du système peut être établi comme dans les travaux de (Sampath, Sengupta, Lafortune, Sinnamohideen, & Teneketzis, 1996) ou (Li, Tong, & Giua, 2020). Ces propositions se basent sur une approche modèle et proposent la construction d'un modèle fautif du système. En ce qui concerne (Sampath et al., 1996) il s'agit de faute non observable, mais dont les conséquences sur le système sont connues. Globalement ces méthodes se basent sur la *signature* de l'attaque.

Une *signature* est un modèle qui correspond à une menace connue (Scarfone, Mell, et al., 2007). La détection basée sur les signatures est le processus qui consiste à comparer les signatures aux événements observés afin d'identifier les incidents possibles. Un exemple de signature est un

e-mail dont l'objet est "Photos gratuites" et le nom de fichier de la pièce jointe est "freepics.exe", qui sont des caractéristiques d'une forme connue de logiciel malveillant.

La détection basée sur les signatures est très efficace pour détecter les menaces connues, mais présente plusieurs limites (Hubballi & Suryanarayanan, 2014). Il est souvent difficile d'écrire des signatures de bonne qualité (Paxson, 1999). Une signature doit être capable de détecter toutes les variations possibles d'une attaque pertinente et ne pas détecter toutes les activités non intrusives. Mais par exemple, si un attaquant modifie le logiciel malveillant de l'exemple précédent pour utiliser le nom de fichier "freepics2.exe", une signature recherchant "freepics.exe" ne correspondra pas.

De plus, l'écriture de signatures dépend fortement des connaissances des experts. Comme de nouvelles failles et vulnérabilités sont découvertes en permanence, pour écrire de bonnes signatures, il faut avoir une compréhension complète du comportement et disposer de suffisamment de données à analyser. En raison de cette dépendance, cette méthode est toujours sujette à des erreurs.

C'est la raison pour laquelle nous allons nous baser sur des techniques de détection basées sur la construction d'un modèle non fautif.

2.3.2 Techniques basées sur un modèle non fautif

La recherche des anomalies repose essentiellement sur deux éléments. Il s'agit d'établir des profils de comportement pour les activités normales et les activités courantes. Ces profils sont ensuite examinés à la lumière de différentes techniques afin d'identifier toute forme de déviation par rapport au comportement normal.

2.3.2.1 Principe de détection

Bien qu'une anomalie soit définie par les chercheurs de diverses manières en fonction de son domaine d'application, une définition largement acceptée est celle d'Hawkins :

An anomaly is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.

Les anomalies indiquent des événements significatifs qui sortent de l'ordinaire pouvant inciter à prendre des mesures critiques dans un large éventail de domaines d'application. La détection d'anomalies est un problème qui peut avoir diverses applications (Muruti, Rahim, & bin Ibrahim, 2018). Une analyse d'anomalies des dossiers médicaux (électroniques) pourrait indiquer la présence d'épidémies ou la recherche d'anomalies dans les données des transactions par carte de crédit pourrait être utilisée pour prouver la fraude par carte de crédit. Par exemple, un modèle de trafic inhabituel dans un réseau pourrait signifier qu'un ordinateur a été piraté et que des données sont transmises à des destinations non autorisées. À titre illustratif, Fig. 2.6 présente un exemple d'anomalies à deux dimensions, les régions N_1 et N_2 représentent des zones avec des données normales, car la majorité des observations se trouvent dans ces zones. Alors que les points O_1, O_2 , et la zone O_3 qui sont plus éloignés des zones normales sont des anomalies.

À partir de cette classification, il y a deux façons de procéder. Tout d'abord les techniques (Chandola et al., 2009) qui visent à déterminer un *score*. Ces techniques attribuent un score d'anomalie à chaque instance des données de test en fonction de la déviation par rapport au modèle nominal. Pour l'exemple de la Fig. 2.6, graphiquement, il est clair que O_1 aura un score plus élevé que pour O_2. Par la suite, la question est, connaissant ce score, comment définir la limite

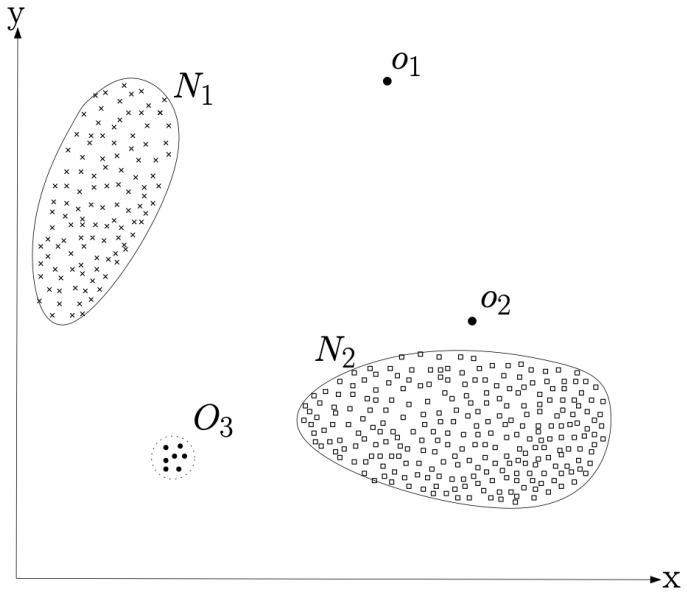


FIGURE 2.6 – Un exemple simple d'anomalies dans un ensemble de données à deux dimensions (Chandola et al., 2009).

définissant une anomalie. C'est-à-dire le seuil définissant une anomalie. Une autre possibilité serait binaire : si l'observation n'est pas dans l'ensemble défini comme nominal alors c'est une anomalie.

Maintenant, la problématique réside dans la construction du modèle. Il existe différentes façons de le construire. Cela peut être par connaissance experte avec la détermination d'une spécification du comportement du système, (Ko, Ruschitzka, & Levitt, 1997), cela est utilisé dans la sécurité de système cyberphysique (Nweke, 2021) ou pour établir la spécification d'un protocole réseau. Par exemple, (Tseng et al., 2003) propose la spécification d'un protocole réseau de routage : Ad hoc On-Demand Distance Vector (AODV). Cette approche utilise des automates finis pour spécifier le comportement correct du routage AODV et de moniteurs de réseau distribués pour détecter les violations des spécifications au moment de l'exécution. Ce n'est pas la seule spécification qu'on retrouve dans la littérature sur ce protocole, par exemple (Uppuluri & Sekar, 2001). Cette solution implique une connaissance experte de l'algorithme. De plus, ce modèle est construit dans l'objectif de détecter certaines attaques en particulier sélectionnées : (Uppuluri & Sekar, 2001) et (Tseng et al., 2003) spécifient le même protocole, mais diffèrent dans les attaques sélectionnées. Ainsi, chaque spécification a de meilleures performances face aux attaques considérées. Ainsi, bien que les autres attaques peuvent être détectées, ce type de construction présente des limites confrontées à une grande diversité d'attaque.

Une autre possibilité est la construction à partir d'un jeu de données. C'est-à-dire identifier le comportement d'un système à partir d'un ensemble d'observation. Il y a plusieurs façons de construire ce modèle, cela peut-être en utilisant des modèles simples comme les automates temporisés classiques comme dans (Fouquet, Faraut, & Lesage, 2020) ou bien en utilisant d'autres algorithmes de type Machine Learning.

2.3.2.2 Intérêt du Machine Learning

Les environnements réseau évoluant rapidement, des variantes d'attaques et de nouvelles attaques apparaissent constamment. Il est donc nécessaire de développer des approches capables de détecter des attaques inconnues. Pour résoudre ces problèmes, les chercheurs ont commencé à se concentrer sur la construction d'algorithme utilisant des méthodes d'apprentissage automatique de type machine learning (H. Liu & Lang, 2019). Les techniques basées sur l'apprentissage automatique peuvent atteindre des niveaux de détection satisfaisants lorsque suffisamment de données d'entraînement sont disponibles et que les modèles d'apprentissage automatique sont suffisamment généralisables pour détecter les variantes et les nouvelles attaques (Nassif, Talib, Nasir, & Dakalbab, 2021). Les données sont les éléments fondamentaux des techniques de détection d'anomalies puisqu'elles contiennent des caractéristiques liées aux comportements du système. Les types de caractéristiques et les méthodes d'extraction de celle-ci diffèrent selon les données, ce qui entraîne des différences dans les modèles d'apprentissage automatique les plus appropriés.

La différence entre les approches réside dans le type de données en entrée et plus précisément si elles sont étiquetées ou non. Les étiquettes associées à une instance de données indiquent si cette instance est normale ou anormale. Il convient de noter que l'obtention de données étiquetées précises et représentatives de tous les types de comportements est souvent d'un coût prohibitif. L'étiquetage est souvent effectué manuellement par un expert humain et, par conséquent, un effort substantiel est nécessaire pour obtenir l'ensemble de données d'entraînement étiquetées. En général, il est plus difficile d'obtenir un ensemble étiqueté d'instances de données anormales couvrant tous les types possibles de comportements anormaux que d'obtenir des étiquettes pour un comportement normal. De plus, le comportement anormal est souvent de nature dynamique, par exemple, de nouveaux types d'anomalies peuvent apparaître, pour lesquelles il n'existe pas de données d'entraînement étiquetées. Dans certains cas, comme la sécurité du trafic aérien, les cas d'anomalie se traduiraient par des événements catastrophiques, et sont donc très rares.

Les algorithmes d'apprentissage non supervisé peuvent apprendre le modèle nominal du réseau et signaler les anomalies sans aucun ensemble de données étiquetées. Ils peuvent détecter de nouveaux types d'intrusions, mais sont très enclins aux fausses alertes positives (A. Ahmad, Harjula, Ylianttila, & Ahmad, 2020). Pour réduire ces faux positifs, un ensemble de données étiquetées est introduit afin de construire un modèle d'apprentissage automatique supervisé (ou semi-supervisé) en apprenant la différence entre un paquet normal et malveillant dans le réseau. Bien que cet étiquetage soit coûteux, il permet d'améliorer les performances de la détection.

Le modèle supervisé peut traiter les attaques connues avec habileté et peut également reconnaître les variations de ces attaques. Il y a différents formalismes utilisés dans la littérature comme l'arbre de décision (DT) (Quinlan, 1986) (Ingre, Yadav, & Soni, 2017). Ce sont des arbres qui classifient les attributs en fonction de leurs valeurs. Ils sont donc principalement utilisés pour des problèmes de type classification (Ahmim, Maglaras, Ferrag, Derdour, & Janicke, 2019). Ensuite, l'algorithme de Supported Vector Machine (SVM) fonctionne sur le principe de la séparation des classes (Cortes & Vapnik, 1995). Fondamentalement, l'objectif est de déterminer les hyperplans de séparation entre les classes de données en maximisant les marges entre les points les plus proches de la classe (Teng, Wu, Zhu, Teng, & Zhang, 2017). De même, la théorie de Bayes (BT) (Domingos & Pazzani, 1997) utilise la probabilité conditionnelle pour déterminer la probabilité d'un événement en fonction de l'ensemble des données (Jabbar, Aluvalu, & Reddy, 2017). L'objectif est d'inférer sur l'état du système à partir des observations (Fouladi, Kayatas, & Anarim, 2016). Une telle théorie peut supporter plusieurs formalismes comme un automate stochastique classique ou le formalisme du modèle de Markov caché (HMM) introduit dans (Baum & Petrie,

1966) (Devarakonda, Pamidi, Kumari, & Govardhan, 2012). Le HMM est un modèle de Markov statistique dans lequel le système est modélisé comme un processus de Markov avec des états cachés. Cependant, il existe un processus observable qui est une conséquence des états cachés. L'évolution du modèle de Markov ne pouvant être observée directement, l'objectif est de déduire les états cachés à partir des observations. Ces inférences permettent de détecter des anomalies dans les applications de cybersécurité comme dans (Chen, Guan, Huang, & Ou, 2016) ou (Holgado, Villagrá, & Vazquez, 2017). Cette approche discrète du formalisme peut être étendue en considérant une approche continue avec un réseau neuronal récurrent (RNN) (Hochreiter & Schmidhuber, 1997) qui est répandu dans la littérature pour les systèmes de détection d'intrusion comme dans (Nayyar, Arora, & Singh, 2020) (Kim, Kim, Thu, & Kim, 2016). (Yin, Zhu, Fei, & He, 2017). En effet, le deep learning est une branche du machine learning qui peut atteindre des performances exceptionnelles dans le domaine de la détection d'intrusion (H. Liu & Lang, 2019) (Z. Ahmad, Shahid Khan, Wai Shiang, Abdullah, & Ahmad, 2021). Comparées aux techniques traditionnelles, les méthodes de deep learning permettent de mieux traiter des données volumineuses. Une caractéristique de ces types de modèles est qu'il s'agit d'une structure profonde, qui contient plusieurs couches cachées contrairement aux modèles classiques. Les études utilisant de tels modèles ont récemment augmenté (Z. Ahmad et al., 2021).

Ces algorithmes sont des outils afin de modéliser le système pour ensuite déterminer un *score* aux observations. Beaucoup de travaux considèrent plusieurs de ces algorithmes dans le but de les comparer comme dans (A. Ahmad et al., 2020) qui compare les performances de SVM, NB ou DT notamment. Cela est utilisé notamment dans les travaux visant à trouver le formalisme optimal, ce qui ne sera pas le cas de cette thèse. En effet, on cherche à démontrer la faisabilité de la détection d'anomalies dans un contrôle de réseau centralisé et l'optimalité, selon certaines métriques définies, est dépendante du cas d'étude considéré. Ici, on s'appuie sur un cas de routage avec différents algorithmes, mais ils sont là uniquement à titre illustratif et la recherche d'optimalité n'est pas généralisable. Ainsi, on va voir les bénéfices de différentes solutions d'apprentissage selon les cas, mais sans chercher la solution optimale.

Ainsi, cette thèse va se concentrer sur les méthodes de détection basées sur des modèles non fautifs et plus particulièrement sur l'intérêt avéré du machine learning qu'on utilisera pour la détection d'anomalies dans des fonctions réseaux avancées, où il n'est pas possible de répondre de manière binaire quant à la présence d'une faute ou attaque, mais que seul un score de vraisemblance semble évaluable. Pour des fonctions moins avancées, nous allons néanmoins voir qu'il est possible de proposer des méthodes moins avancées que les solutions de machine learning.

2.4 Première solution de détection d'anomalies

Dans cette thèse, nous allons donc construire un modèle non fautif du système afin d'analyser les potentielles déviations, et donc les anomalies, du contrôle. Il existe diverses façons de construire ce modèle et dans un premier temps nous allons proposer une spécification de l'activité de la commande et c'est l'objet de cette section. Il s'agit en fait de construire une spécification de l'activité de la commande, c'est-à-dire un modèle basé sur notre connaissance experte du fonctionnement de l'algorithme.

2.4.1 Définition du *template*

Pour développer la logique de détection, la théorie des systèmes de détection d'intrusion (IDS) a été prise en compte. Selon (Liao, Lin, Lin, & Tung, 2013b), les propositions d'IDS peuvent être

divisées en deux catégories : se concentrer sur le comportement d'attaque ou sur le comportement non défectueux du système comme vu ci-dessus. Pour rappel, la première approche est basée sur la signature de l'attaque. Cela implique de prendre en compte des attaques particulières et de ne détecter que celles qui sont prises en compte.

La seconde est divisée en approches basées sur les anomalies et les spécifications. La première approche vise à construire le modèle à partir d'un jeu de données et la seconde à partir de connaissance experte, mais toutes deux visent à comparer le comportement non défectueux connu au comportement en cours. Fondamentalement, les techniques de détection des anomalies sont basées sur un modèle du comportement défectueux du système, tandis que les techniques basées sur les spécifications sont basées sur une spécification directement issue de la documentation.

Ces deux techniques peuvent être combinées comme proposé dans (Sekar et al., 2002). Il y a deux étapes : une spécification est déterminée hors ligne, puis le système est observé en ligne pour déterminer un modèle de son comportement récurrent et apprendre certaines propriétés statistiques liées au modèle de spécification. Dans ce travail, une approche similaire est introduite. Une spécification du comportement de l'activité de la commande sera établie et ensuite on déterminera un modèle statistique des décisions prises. Le formalisme de spécification choisi est un *template*, inspiré de (Pandalai & Holloway, 2000), qui exprime la causalité entre plusieurs événements. En particulier, un *template* est composé de deux événements : un premier, l'événement déclencheur, qui lors de son occurrence donne l'information qu'un second événement, celui attendu, devra se produire dans un certain intervalle de temps. Par exemple, dans un réseau nominal lorsqu'un terminal injecte du trafic en direction d'un second alors, ce second terminal va le recevoir. Ici, l'événement déclencheur est la requête du commutateur et l'événement attendu est l'action dictée par le contrôleur. L'intervalle de temps de l'événement attendu sera à déterminer, et est notamment fonction de la latence du réseau.

2.4.2 *Template* de l'activité de la commande

Ici, par rapport à notre problème, un *template* exprime la causalité entre les requêtes et les commandes. Cette spécification évolue en fonction des variables internes du contrôleur qui sont estimées par l'observation de l'activité de la commande. Formellement, considérons $\mathcal{T}$ l'ensemble des *templates* et un *template* $temp \in \mathcal{T}$ est défini et instancié à notre problème comme :

- $temp = \{cond, \{(cmd_i)_{i\in[1,n]}, it\}\}$
- $cond \in \mathcal{I} \cup \mathcal{P} \cup \mathcal{S}$: l'événement déclencheur. Cela peut être une requête d'un switch ou un lien cassé.
- $(cmd_i)_{i\in[1,n]} \in \mathcal{A}^n$: un ensemble d'actions attendues en réaction à une requête.
- $it \in \mathbb{R}^2$: l'intervalle de temps dans lequel la commande est attendue.

Plus précisément, un *template* $temp$ spécifie qu'après la capture d'une requête provenant d'un commutateur $cond$, l'observateur attend un ensemble de commandes correspondant à la mise en place de la route (ou plus généralement du plan de données) sur le réseau $(cmd_i)_{i\in[1,n]}$. De plus, on s'attend à ce que ces commandes soient envoyées sur le réseau dans un intervalle de temps it. Une illustration d'un template est représentée en Fig. 2.7. Ce template représente la spécification du comportement du contrôleur. Le comportement spécifié ici est que le contrôleur doit mettre en place, dans un certain intervalle de temps, un plan de données en réponse à une requête. Le contrôleur devra donc respecter ce *template*, sinon cela correspondra à une anomalie dans l'activité de la commande. Par exemple, en cas de défaillance du contrôleur, les commutateurs n'obtiendront pas de réponse à leurs requêtes ce qui correspond à une infraction de la spécification puisque l'observateur n'aura pas de commande dans l'intervalle de temps prévu.

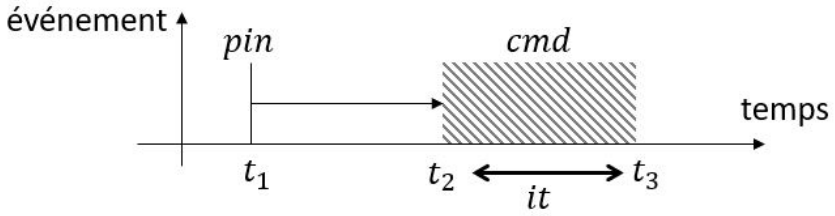


FIGURE 2.7 – Un exemple de *template*.

Cette spécification permet de vérifier la bonne forme de l'activité de la commande. C'est-à-dire que le contrôleur met en place un plan de données lorsque c'est nécessaire. Ensuite, il faut vérifier que le plan de données en place ne présente pas d'anomalies. Pour cela, on propose de comparer ce plan de données à un modèle du comportement non fautif du contrôle préalablement établi. Ce modèle portera sur les variables internes du contrôleur et l'objectif est de les re-estimer, dans un cas supposé nominal, puis de suivre leurs évolutions. En effet, le plan de données attendu évolue en fonction de ces variables internes. Dans le cas d'un contrôle de type commutation, les décisions du contrôleur évoluent en fonction de sa table de routage. Ainsi, lorsque le contrôleur reçoit une requête pour un flux (`Packet_In`) il doit prendre une décision (`Flow_Mod`) en fonction de ses tables de routage afin d'orienter le flux. Finalement, le principe est le même que dans un réseau classique où les commutateurs prennent des décisions selon leurs tables de routage si ce n'est que, dans le cas d'une architecture SDN, les tables de routages sont maintenues par le seul contrôleur.

Néanmoins, sans interface est-ouest, il n'y a pas d'accès à ces variables par l'observateur. Par conséquent, l'observateur va devoir estimer les variables internes à partir des seules décisions prises par le contrôleur. Dans un premier temps, un modèle des états internes du contrôleur va être établi lors d'une phase d'apprentissage en utilisant les décisions prises par le contrôleur. À noter que cette phrase n'est pas statique hors ligne, mais il y a un renforcement permanent. Dans un deuxième temps, il faudra suivre l'évolution de ces variables internes afin de vérifier qu'il n'y a pas d'anomalies. S'il n'y en a pas alors l'observateur pourra réutiliser ces informations pour renforcer le modèle statistique.

Par exemple, l'observateur pourra estimer les tables de routage du contrôleur à partir de ses décisions : si le contrôleur ordonne de faire passer le flux par un port particulier pour une infrastructure alors l'observateur pourra inférer que la variable interne du contrôleur concerné, ici la table de routage pour des commutateurs, comporte l'information que pour rejoindre la destination de ce flux il faut passer par ce port. Ici, ce n'est pas la pertinence du contrôle qui est jugée, mais sa cohérence. On ne cherche pas à reproduire l'algorithme de contrôle pour vérifier qu'on obtient le même résultat puisque ce n'est possible qu'en considérant des algorithmes déterministes. On va donc établir une estimation des variables internes du contrôleur pour suivre leurs évolutions et vérifier que c'est vraisemblable. En conséquence, nous faisons l'hypothèse que le comportement du contrôleur est sans anomalies lors de la phase d'apprentissage. Effectivement, si le contrôleur est attaqué avant que l'observation n'ait commencé, alors le comportement malveillant sera considéré comme la référence.

En conclusion, la première étape est de vérifier que l'activité du contrôle correspond au *template* de l'algorithme de commande. Ce *template* représente seulement la forme de l'activité de la commande. Dans un second temps, il est nécessaire d'établir un modèle non fautif de l'activité de la commande lors d'une phase d'apprentissage pour vérifier la vraisemblance de ce que l'on observe. Les outils en lien avec cette deuxième étape seront développés dans le chapitre

suivant. Dans la suite nous allons présenter un outil de projection qui permettra d'obtenir cette spécification à partir d'un algorithme de commande.

La méthode de détection peut donc être représentée symboliquement par l'automate donné en Fig. 2.8.

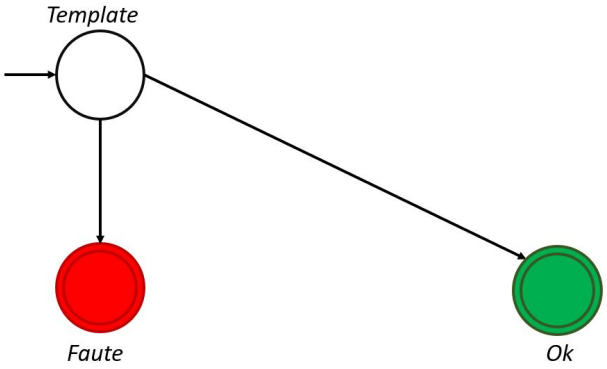


FIGURE 2.8 – Modèle de notre méthode de détection en ne considérant que la spécification de l'activité du contrôle sous forme de *template*.

À l'observation d'une requête, l'observateur vérifie que le contrôleur respecte la spécification du contrôle correspondant au *template*. C'est-à-dire qu'il répond à la requête dans les temps. Ceci n'est toutefois pas suffisant pour assurer que le contrôle est totalement sans anomalies. En effet, les commandes peuvent être malveillantes comme développé dans les chapitres précédents.

2.4.3 Projection

Nous supposons que l'algorithme de contrôle est connu et à partir de cette connaissance nous introduisons une fonction de projection qui vise à déterminer le modèle de spécification des algorithmes de commande tels que le routage de Bellman Ford par exemple, le filtrage de paquets, etc.

La causalité action/réaction de l'algorithme est située dans les instructions de ces algorithmes. En effet, la sortie de l'algorithme correspond à un ensemble de n commandes $cmd_i \in \mathcal{A}^n$. Cet ensemble est retourné si la condition de l'instruction est vérifiée. Cette condition peut être une requête d'une infrastructure $pin \in \mathcal{I}$, la notification du statut d'un port indiquant que le lien d'un chemin est cassé $ps \in \mathcal{N}$ ou bien une statistique montrant une augmentation du trafic par exemple $stats \in \mathcal{S}$. Cette condition dépend des algorithmes de contrôle. Par conséquent, nous avons décidé de modéliser un algorithme de commande $Algo$ par ces m instructions $istr$.

$$Algo = \mathbb{I}^m \text{ with } \mathbb{I} = (\mathcal{I} \cup \mathcal{N} \cup \mathcal{S})^{\mathbb{B}} \times Op \times \mathcal{A}$$

De manière simplifiée, une instruction $istr \in \mathbb{I}$ est définie par :

— $istr = (cond, op, cmd)$
— $cond \in (\mathcal{I} \cup \mathcal{N} \cup \mathcal{S})^{\mathbb{B}}$: une équation booléenne validée par une requête, une évolution du statut d'un port ou une statistique.
— $op \in Op$: un ensemble d'opérations nécessitant un certain temps de calcul. Le développement est présenté ci-après.
— $cmds = \cup_{i=1}^{n} cmd_i \in \mathcal{A}^n$: la sortie correspondant aux commandes envoyées aux infrastructures.

Chaque instruction est composée par un ensemble d'opérations nécessitant un certain intervalle de temps. Dans cette section, nous simplifierons la façon de déterminer cet intervalle de temps en introduisant une fonction Est. Cette fonction associe à toute instruction $istr$ l'intervalle de temps it nécessaire à son exécution de sorte que :

$$Est(istr) = it \in \mathbb{R}^2$$

Le lien entre l'algorithme de commande et le *template* est clair. L'élément déclencheur du *template* est l'élément qui vérifie la condition d'une instruction de l'algorithme. Par la suite, le *template* laisse donc un intervalle de temps it, lié au temps d'effectuer les opérations de l'instruction, pour attendre les commandes cmd, à savoir la sortie de l'instruction. Nous le formaliserons en utilisant la fonction de projection $Proj$ définie telle que :

$$Proj(Algo) = \bigcup_{istr=(cond,op,cmds)\in Algo} \{cond, \{\cup_{i=1}^{n} cmd_i, Est(istr)\}\}$$

Cette projection nous permet d'obtenir la spécification du comportement du contrôleur qui correspond à l'ensemble des *templates* de l'activité de la commande. C'est-à-dire que le contrôleur répond dans les temps pour chaque requête par un ensemble de commandes attendu. Sinon, cela correspondra à une anomalie qui peut être liée à une défaillance du contrôleur.

2.4.4 Propriétés temporelles

Ce *template* se base sur une borne temporelle. Dans cette partie, nous présentons comment déterminer les bornes de l'intervalle it correspondant au temps d'attente des commandes de la part de l'observateur. Deux approches vont être présentées, une première visant à déterminer un intervalle de confiance du temps de réponse et une deuxième visant à déterminer un majorant du temps de réponse du contrôleur. Dans les deux approches un ensemble de n mesures sera utilisé.

Intervalle de confiance

L'idée de cette approche est de déterminer l'intervalle du temps de réponse du contrôleur sous la forme d'un intervalle de confiance. Cet intervalle de confiance permet d'encadrer la valeur réelle du temps de réponse du contrôleur que l'on cherche à estimer. Cela permet de définir une marge d'erreur entre les résultats des n premières mesures et ce qu'on va observer. Par ailleurs, un intervalle de confiance doit être associé à un niveau de confiance.

Afin de garder une réactivité au réseau, on ne va considérer qu'un horizon limité est donc qu'un petit nombre de mesures. Ainsi, on propose de déterminer l'intervalle de confiance en utilisant la loi de Student puisqu'on considère un petit nombre de mesures et cette loi est tout indiquée dans ce cas de figure. Pour ce faire, il faut déterminer la valeur moyenne t_{Moy} de notre échantillon de n mesures :

$$t_{Moy} = \frac{\Sigma_{i=1}^{n} t_i}{n}$$

Ensuite, pour trouver l'incertitude sur cette moyenne, on calcule d'abord l'écart-type σ_{n-1} de la distribution des n mesures :

$$\sigma_{n-1} = \sqrt{\frac{\Sigma_{i=1}^{n} (t_i - t_{Moy})^2}{n-1}}$$

Enfin, on calcule l'écart-type σ_x sur la moyenne qui sera l'incertitude type : $\sigma_x = \sigma_{n-1}/\sqrt{n}$. Cependant, étant donné qu'on ne considère qu'un petit nombre de mesures, on utilisera la méthode de Student. Elle consiste à coefficienter, par k, l'écart-type sur la moyenne en fonction du nombre de mesures effectuées :

$$\Delta t_{Moy} = k \times \frac{\sigma_{n-1}}{\sqrt{n}}$$

La fixation de k est très importante puisqu'elle fixe la tolérance sur le temps de réponse du contrôleur et dépend de deux paramètres : la précision souhaitée et le nombre n de mesures à disposition.

En conséquence, l'intervalle de confiance it est déterminé de la façon suivante :

$$it = [t_{Moy} - \Delta t_{Moy}, t_{Moy} + \Delta t_{Moy}]$$

Cette méthode permet de déterminer expérimentalement l'intervalle it correspondant au temps d'attente des réponses du contrôleur d'après le *template* du comportement de la commande. Il s'agit également de la résultante de la fonction estimation Est introduite lors de la définition de la fonction de Projection.

Cela apporte également une certaine flexibilité/tolérance lors de la fixation de l'intervalle, mais présente des limites. En effet, étant donné que le but est de déterminer un intervalle de confiance à $X\%$, alors il est clair que cette méthode assume un certain nombre de fausses alarmes (ce qui peut être dommageable). C'est pour cela que nous allons proposer une deuxième approche visant à établir un majorant du temps de réponse du contrôleur.

Majoration du temps de réponse par étude du pire cas

L'objectif est toujours d'introduire une tolérance pour le temps de réponse du contrôleur par la définition d'une limite de temps. Cette protection ne doit pas être fixe, mais doit évoluer en temps réel pour suivre l'évolution du trafic et le contexte. En effet, le temps de réponse du contrôleur évolue au fur et à mesure du temps pour plusieurs raisons en lien avec le contexte comme une augmentation du nombre de requêtes provenant des infrastructures.

Pour ce faire, les temps de réponse du contrôleur sont enregistrés dans une liste Lt_{Rep} qui est mise à jour à chaque nouvelle observation. La longueur de cette liste est fixée de sorte à limiter le contexte des valeurs observées et ne retenir que ce qui est adapté à la situation et ne pas être polluée par un autre contexte (tel qu'une augmentation des requêtes des commutateurs ou une absence de requête, par exemple).

Par conséquent, pour déterminer la valeur de la frontière, nous proposons d'analyser l'ensemble des valeurs précédentes et de définir une tolérance sur le pire cas observé. Cela correspond à une protection face à l'incertitude du retard du contrôleur. Nous introduisons donc dans un premier temps le pire cas observé t_{WC}, comme suit :

$$t_{WC} \in Lt_{Rep} | t_{WC} = \max_{t \in Lt_{Rep}} t$$

Sur cette base, la valeur limite t_{borne} est calculée comme étant une proportion de ce pire cas (afin de tenir compte de la rareté de cet évènement et donc la potentielle incomplétude de la liste). Le facteur de protection est $\beta \in \mathbb{R}^+$, permet d'étendre la tolérance sur le temps d'attente du contrôleur. Il a également un impact sur la réactivité du contrôleur comme on le verra dans le chapitre suivant.

$$t_{borne} = \beta \times t_{WC}$$

Ce paramètre permet de fixer une incertitude sur le potentiel retard du contrôleur. Ce retard peut être dû à une attaque DDoS ou une défaillance. Cependant, si cette attaque a un impact limité qui permet au contrôleur de réaliser sa tâche dans la limite, alors aucune faute n'est déclarée par l'observateur, car l'activité de la commande est satisfaite puisqu'elle respecte la tolérance. Ce sera typiquement observable en cas d'attaque DDoS douce visant à ralentir le contrôleur de manière légère en provoquant seulement un petit retard par rapport à la fois précédente (Tripathi & Hubballi, 2018) (Cambiaso, Chiola, & Aiello, 2019) (de Miranda Rios, Inácio, Magoni, & Freire, 2021). Le risque de notre méthode est donc de considérer un facteur de protection supérieur à l'évolution du temps de réponse du contrôleur. En conséquence, puisque ce retard sera dans la limite, il sera alors ajouté à Lt_{Rep}, cela aura pour effet de décaler la borne. De manière itérative, la borne va se décaler petit à petit et tous les prochains retards du contrôleur seront donc dans la limite, jusqu'à potentiellement $+\infty$. Pour pallier cet inconvénient, nous proposons d'introduire une contrainte formalisée comme une limite arbitrairement fixée t_{lim}, qui correspond à une contrainte utilisateur/opérateur, définie par l'application de commande, de l'évolution de ce temps limite.

Finalement, la borne temporelle *timeout* est la borne calculée sauf si elle dépasse la contrainte applicative définie par l'application de commande, c'est-à-dire en prenant la valeur minimum entre la borne calculée et la limite arbitraire comme suit :

$$timeout = \min(t_{lim}, t_{borne})$$

Cette condition temporelle est nécessaire, mais non suffisante puisque les réponses peuvent présenter des anomalies. Dans le chapitre suivant, nous allons donner des conditions nécessaires, sous forme de propriétés structurelles, à l'absence d'anomalie dans la réponse du contrôleur.

Par ailleurs, ici on ne détermine que la borne supérieure du temps de réponse du contrôleur et non un intervalle de ce temps de réponse. En effet, une réponse du contrôleur avant qu'il ait eu le temps de faire le calcul est une anomalie et peut traduire la présence d'un attaquant qui a pris possession du contrôleur. Cependant, les réponses du contrôleur sont analysées et on ne cherche pas à évaluer l'état du contrôleur, mais l'état de l'activité de la commande. Donc si un attaquant permet d'assurer un contrôle cohérent et vraisemblable plus rapidement que le ferait le contrôleur nominal, alors aucun problème n'est considéré.

2.5 Proposition d'un premier observateur [Desgeorges *et al.*, CESCIT 2021]

Dans cette section nous allons présenter la méthode de détection sur un cas simple. Le contrôleur est ici seulement en charge de la commutation de niveau 2. C'est-à-dire que pour chaque requête, il ordonne de retransmettre sur le port correspondant à sa table de commutation. De plus, on considérera que l'algorithme de contrôle est déterministe.

2.5.1 Détection

Comme mentionné et développé plus haut, le principe de la méthode est de vérifier que le comportement du contrôleur respecte le *template* déterminé. La première étape consiste donc à déterminer le *template*. L'algorithme de commande est de la commutation de niveau 2 donc à la réception d'un paquet le contrôleur a deux comportements possibles :

— Il possède l'adresse destination dans sa table MAC : il ordonne au switch de transmettre le paquet au port inscrit dans sa table MAC.
— Il ne connaît pas l'adresse destination : il ordonne au switch d'inonder le paquet.

L'algorithme est donc structuré en deux instructions : $istr_1$ dans le cas où le contrôleur possède l'adresse MAC de la destination dans sa table alors la commande correspond à la retransmission du paquet sur un port particulier et $istr_2$ dans le cas où l'adresse destination est inconnue par le contrôleur alors il va ordonner l'inondation du paquet. Ces commandes dépendent des variables internes du contrôleur, mais nous n'y avons pas accès. Nous allons donc devoir estimer ces variables internes, ici la table MAC, notée MTN, du contrôleur. Pour rappel, on se place dans un contexte SDN où le contrôleur est chargé du contrôle. En effet, dans un réseau classique régi par de la commutation de niveau 2 chaque switch tient à jour sa propre table MAC alors que dans un réseau SDN c'est le contrôleur qui tient à jour symboliquement les tables MAC de chaque switch. On définit donc la table MAC estimée de la façon suivante $MTN = \{(S, MT)\}$ avec :

— S : le commutateur considéré.
— $MT = \{(p, MAC)\}$: la table MAC du commutateur S avec p le port pour joindre l'adresse MAC MAC.

Le processus de détection est donné en Fig. 2.9.

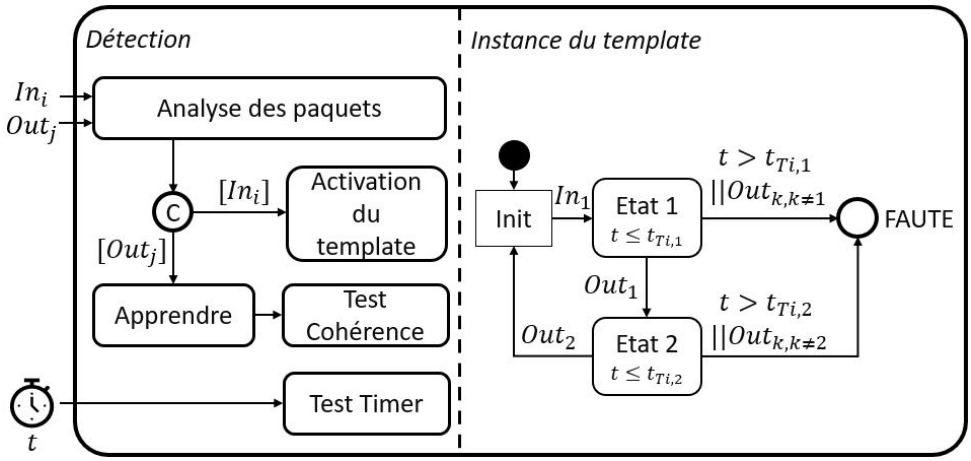


FIGURE 2.9 – Représentation de la méthode de détection

L'entrée de l'algorithme de détection est l'activité de la commande, donc la première étape est de traiter ces paquets par Algo. 1 qui lance l'algorithme lié au paquet observé. Il correspond à la jonction de la Fig. 2.9.

Algorithm 1: Traitement d'un nouveau paquet

Input: Paquet p, Table MAC Estimée MTN et O
if $p \in \Sigma_{In}$ **then**
  $Instance(p, MTN, O)$;
else
  $Match(p, MTN, O)$;

Dans le cas d'un `Packet_In`, Algo. 2 est lancé et le *template* est instancié. Les commandes at-

tendues sont obtenues à partir des estimations des variables internes du contrôleur, ici par MTN. Cela correspond à lancer l'instance du *template* comme représentée dans la Fig. 2.9. L'ensemble des instanciations du modèle à une date t est noté $O(t)$. On fait l'hypothèse qu'initialement il n'y a pas de commande attendue : $O(t = 0) = \emptyset$.

Algorithm 2: Lancement du *template*

Input: *Packet_In pin*, Table MAC Estimée MTN et O
$i = 1$, $Know = 0$, $MTN_{Check} = MTN[1]$;
$Temp[1] = pin$, $Temp[2][2] = it$;
while $MTN_{Check}[1]! = pin[1][2]$ **do**
  $MTN_{Check} = MTN[i + 1]$, $i = i + 1$;
$MT = MTN_{Check}[2]$;
for $Dest \in MT[2]$ **do**
  if $Dest = pin[3]$ **then**
    $Temp[2][1] = MT[1]$, $Know = 1$;
    $O = O \cup Temp$, $Timer(Temp)$;
if $Know = 0$ **then**
  $Temp[2][1] = \emptyset$,$O = O \cup Temp$,$Timer(Temp)$;

Dès que le modèle est instancié, une temporisation est lancée par Algo. 3 pour vérifier que la commande est dans les temps. C'est-à-dire vérifier la partie temporelle du *template*. Cela consiste à attendre la dernière action avant de retirer l'instance de O.

Algorithm 3: Mise en place d'une temporisation

Input: Une instance $Temp$
$fault = False$, $sleep(min(Temp[2][2]))$;
if $Temp \notin O$ **then**
  $fault = True$;
$sleep(max(Temp[2][2]) - min(Temp[2][2]))$;
if $Temp \in O$ **then**
  $fault = True$;
return $fault$;

En parallèle, lorsqu'une commande est observée, la cohérence de la commande est vérifiée par Algo. 4 comme représenté dans la Fig. 2.9. Pour chaque instance, il y a une vérification que la commande était celle attendue par cette instance. Globalement, ici, les actions attendues correspondent à ce qui est dans la table MAC estimée. Si elle était attendue alors l'instance est supprimée de O. Sinon, la commande n'était pas attendue et on suppose qu'elle est incohérente. De plus, lorsqu'une nouvelle commande est observée, l'observateur apprend cette commande et il est nécessaire de mettre à jour l'estimation des variables internes du contrôleur par Algo. 5 (forme de renforcement de l'observateur).

Mettons en pratique cette méthode sur un cas simple.

Algorithm 4: Vérification de la cohérence des commandes observées

```
   Input: Packet_ Out pout, Table MAC Estimée MTN et O
 2 fault = True , Learn = 0 ;
 3 for Temp ∈ O do
 4     if Temp[2][1] = pout then
 5         fault = False , O = O \ Temp ;
 6     else if Temp[2][1] = ∅ and pout[3] = Temp[1][1][2] then
 7         fault = False , O = O \ Temp , Learn = 1 ;
 8 if Learn = 1 then
 9     Learn(MTN, pout) ;
10 return fault ;
```

Algorithm 5: Algorithme d'apprentissage (estimation des variables internes).

```
   Input: Packet_ Out pout et la Table MAC Estimée MTN
 2 MTN_Check = MTN[1] , newLigne = 1 , i = 1 ;
 3 while MTN_Check[1]! = pout[3] do
 4     MTN_Check = MTN[i + 1] , i = i + 1 ;
 5 MT = MTN_Check[2] ;
 6 for Dest ∈ MT[2] do
 7     if Dest = pout[5] then
 8         newLigne = 0 ;
 9 if newLigne = 1 then
10     ML_new = (pout[5], pout[1]) ;
11     MT[2] = MT[2] ∪ ML_new;
```

2.5.2 Scénario

La topologie représentée sur la Fig. 2.10 est simulée à l'aide de Mininet [1] (De Oliveira, Schweitzer, Shinoda, & Prete, 2014) qui est un émulateur de réseau SDN. Un contrôleur ONOS [2] (ONF, 2017) est chargé avec l'application de commutation de niveau 2 (apprentissage MAC). Les paquets seront capturés en utilisant Wireshark [3], un outil de capture de paquet.

Dans les futures captures Wireshark l'identifiant d'un switch sera différent de celui en Fig. 2.10. Dans la Table. 2.1 nous donnons l'identifiant des commutateurs de Fig. 2.10 selon la trame Wireshark considérée. Par exemple, le commutateur identifié par $of01$ dans la Fig. 2.10, sera identifié par le numéro 121 dans la trame Wireshark donnée en Fig. 2.11.

Le scénario appliqué est un ping de $h1$ à $h2$. La méthode de détection sera mise en pratique dans trois cas : sans anomalies, avec une attaque et avec une défaillance. Les trames, capturées à l'aide de Wireshark, du scénario sans anomalies, sont représentées sur la Fig. 2.11. Pour faciliter la lecture, la valeur des instances du modèle et l'évolution de O sont détaillées dans la Fig. 2.11. Il en sera de même pour le cas d'attaque et le cas de défaillance.

Nous ne développerons pas le calcul des contraintes temporelles pour la démonstration et on

1. `http://mininet.org/`
2. `https://opennetworking.org/onos/`
3. `https://www.wireshark.org/`

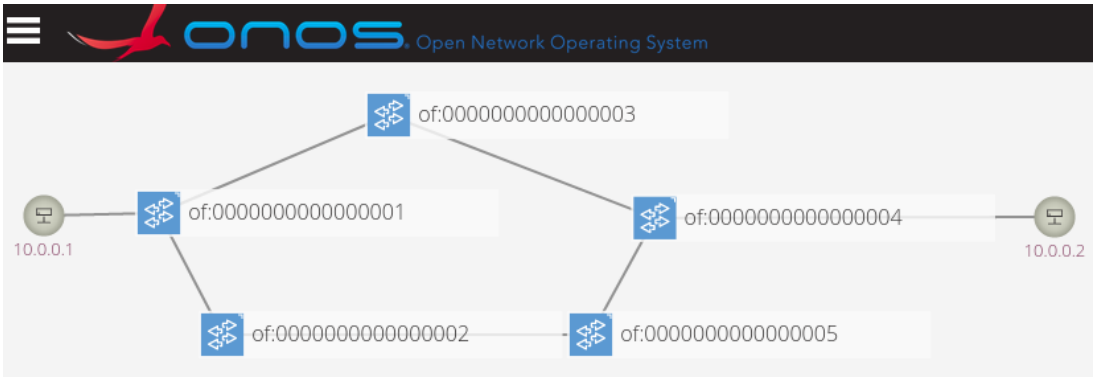


FIGURE 2.10 – La topologie du réseau.

Topologie	Fig. 2.11	Fig. 2.12	Fig. 2.13
$of01$	121	1	2
$of02$	124	2	4
$of03$	122	0	3
$of04$	120	3	1
$of05$	123	4	5

TABLE 2.1 – Identifiant des commutateurs de la topologie en Fig. 2.10.

fixera : $Est(istr_1) = Est(istr_2) = [0.0005, 0.02]$ secondes $= [0.5, 20]$ ms. Donc, la projection de l'algorithme donne :

$$Proj(Algo) = \{cond_1, \{pout_1, [0.5, 20]\}\} \bigcup \{cond_2, \{pout_2, [0.5, 20]\}\}$$

Le *template* est clair : à l'observation d'une requête d'une infrastructure il est attendu que le contrôleur répondent par une seule commande et pour cela il a entre $0,5$ et 20 millisecondes.

L'évolution de $O(t)$ concernant le switch 4 est détaillée dans la Fig. 2.11. Tout d'abord, le Packet_In pin_{2166} est observé. L'algorithme principal Algo. 1 lance l'instanciation par Algo. 2 afin de compléter O. La condition ligne 11 n'est pas remplie, car pin_{2166} n'a pas encore été observé, donc aucune commande spécifique n'est attendue ainsi le template est complété avec l'attente de n'importe quelle commande. On est dans la phase d'apprentissage. La commande $pout_{2170}$ est observée à destination de switch 4. De même, l'Algo. 1 lance la vérification par Algo. 4. Comme une instance attend $\emptyset$ pour le switch 4, la condition ligne 6 de l'Algo. 4 est remplie et l'instance sera supprimée de O. De plus, Algo. 5 est lancé pour enregistrer la commande dans MTN comme représenté dans la Fig. 2.11.

Cette phase d'apprentissage sera similaire pour pin_{2174}. L'observateur va apprendre que le switch s_4 commute sur le port 1 les flux à destination de h_1.

Ensuite, à l'observation des requêtes déjà vues, comme pin_{2199}, l'instanciation du *template* est faite à partir de l'estimation de la table MAC du contrôleur MTN, selon la condition ligne 11 d'Algo. 6. Par la suite, les commandes $pout_{2200}$ sont observées et sont similaires à celle apprise préalablement $pout_{2182}$. En conséquence, la condition ligne 4 d'Algo. 4 est remplie, l'instance est supprimée de O et aucune anomalie n'est détectée.

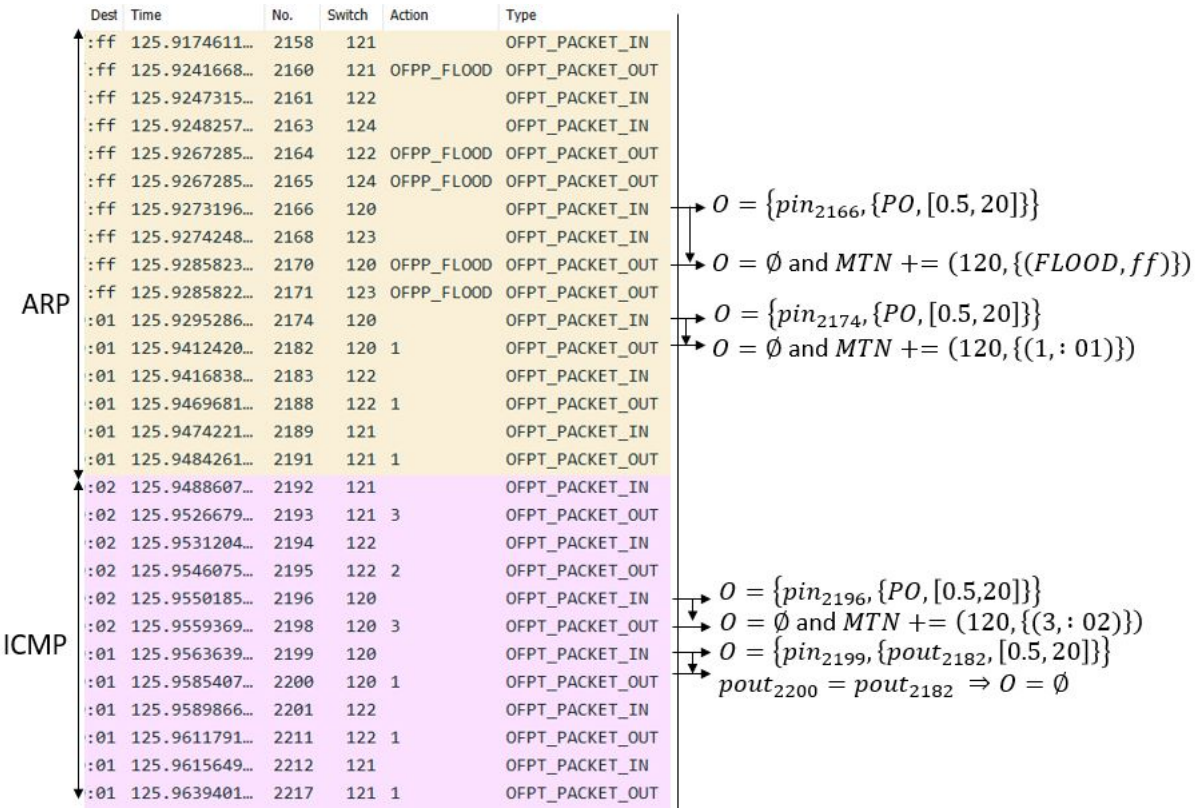


FIGURE 2.11 – Paquets échangés lors d'un ping entre h_1 et h_2

2.5.3 Cas d'attaque

Considérons l'attaque développée dans (P. Porras et al., 2012). Pour modéliser une telle attaque, on introduit un biais b_{Cmd} d'une commande $pout \in \mathcal{A}$ qui conduit à $pout' \in \mathcal{A}$:

$\forall i \in [1, lengh(pout)]$ $pout'[i] = pout[i]$
$pout'[1] = 10$ si $pout[3] = 4$, $pout'[1] = pout[1]$ sinon

L'attaque est faite de sorte que tout le trafic qui passe par le switch 4 est en pratique retransmis sur le port 10. Cela permet de retransmettre le trafic afin de permettre à l'attaquant d'empêcher la transmission ou encore d'inonder l'équipement connecté à ce port. Dans ce scénario le trafic correspond à deux pings : un premier qui permet au détecteur d'apprendre le comportement non défectueux de la commande et un second sous attaque. Les trames correspondantes sont représentées sur la Fig. 2.12.

Le but de ce qui suit est de montrer comment une anomalie est détectée. Tout d'abord, il y a la phase d'apprentissage avec l'observation de pin_{149} et $pout_{151}$ ainsi que de pin_{152} et $pout_{153}$. Sur la base de ces échanges de paquets, l'estimation de MTN a été réalisée par Algo. 5. Par conséquent, lors de l'observation de pin_{327}, le modèle est instancié par Algo. 2 avec l'attente de la même commande que $pout_{139}$. Mais $pout_{329}$ est différent de la commande contenue dans l'instance de O. Ainsi, la condition ligne 25 d'Algo. 4 n'est pas remplie et une alarme est levée comme le montre la Fig. 2.12. Dans cet exemple, le biais considéré était une modification du port de transmission, mais tout autre biais de la commande qui peut être formalisée plus haut serait détecté de manière similaire.

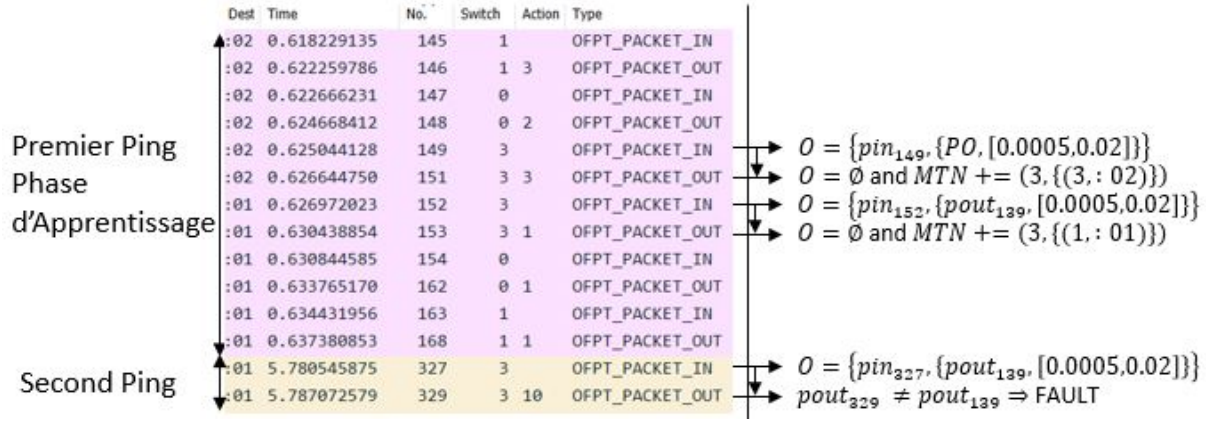


FIGURE 2.12 – Paquets échangés lors de deux pings. Le premier lors de la phase d'apprentissage et le second sous attaque.

2.5.4 Cas de défaillance

Le cas de la défaillance du contrôleur est considéré. Cela se traduit par une commande $pout \in \mathcal{A}$ biaisée $pout' \in \mathcal{A}$ comme suit :

$pout' = \emptyset$

Les trames observées sont représentées dans la Fig. 2.13. À l'observation de pin_{589} du commutateur 4, le modèle est instancié dans Algo. 2 et en parallèle une temporisation est lancée par Algo. 3. Après l'attente de 0,02 seconde, en accord avec le *template*, une faute est déclarée dans Algo. 3 comme représenté sur la Fig. 2.13.

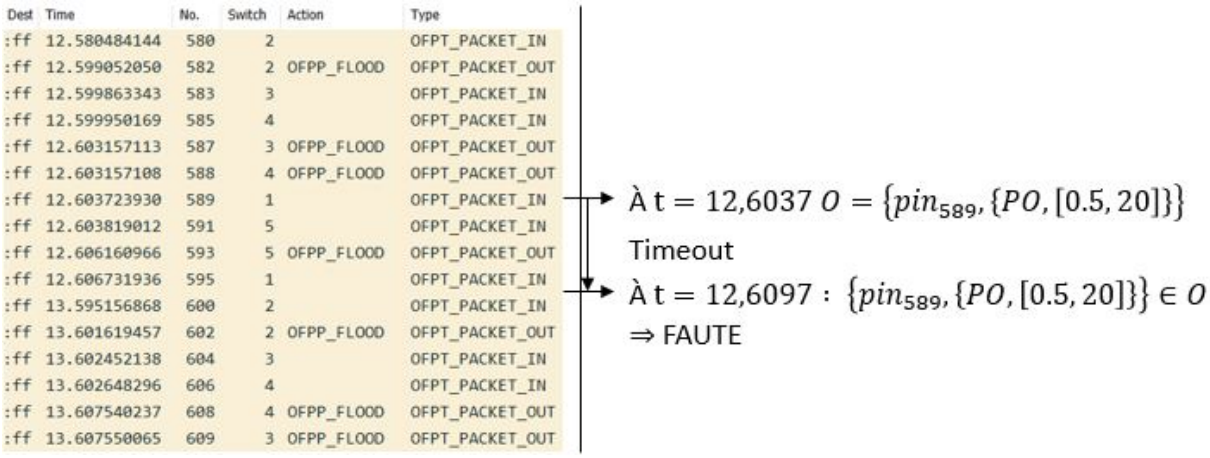


FIGURE 2.13 – Paquets échangés lors d'un ping lors d'un cas de défaillance.

Il est important de constater que la défaillance est détectée uniquement parce que les commutateurs ont envoyé une requête. Sans cela, l'observateur ne se serait pas rendu compte de la défaillance du contrôleur puisque dans ce cas le contrôle ne serait pas impacté. En effet, nous avons fait l'hypothèse que si le contrôleur est défaillant, mais que le réseau n'a pas besoin de lui alors ce n'est pas un souci. Ce qui signifie aussi que si .le contrôleur défaille que pendant un court intervalle de temps durant lequel il n'est pas sollicité, comme dans Fig. 1.11b, alors cette défaillance ne sera pas détectée par l'observateur.

Par ailleurs, cette proposition est limitée puisque ne prennent en considération que les requêtes des commutateurs. L'observateur ne vérifie pas le comportement du contrôleur face à une cassure de lien entre des commutateurs pouvant entraver la transmission de paquet sur le réseau.

2.6 Conclusion

Ce chapitre introduit l'architecture multicontrôleurs proposée dans le but de proposer une solution aux menaces de sécurité et de sûreté de la couche contrôle d'une architecture SDN. Cette architecture se compose d'un contrôleur nominal responsable du contrôle sur le réseau et d'un observateur chargé de détecter toutes anomalies dans les décisions prises par le contrôleur. Cependant, la spécificité de notre architecture de contrôle est l'absence d'interface est-ouest, c'est-à-dire l'interface de communication entre les contrôleurs. En conséquence la détection est basée uniquement sur l'activité de la commande, c'est-à-dire les paquets OpenFlow échangés au niveau de l'interface sud, entre le contrôleur et les commutateurs. Un état de l'art des méthodes de détection d'anomalies a été présenté afin de positionner nos travaux. Dans cette thèse, nous n'avons aucune hypothèse sur le comportement fautif du contrôleur. Ainsi, un modèle non fautif de l'activité de la commande sera établi. Ce modèle peut être construit de deux façons : par connaissance experte ou bien en se basant sur un jeu de données. Ces deux approches vont être combinées. Tout d'abord, une spécification du comportement de l'activité de la commande est déterminée. Nous avons proposé une définition de cette spécification inspirée de la notion de *template* définis par (Pandalai & Holloway, 2000). L'idée est de vérifier que, lorsqu'il a une requête, le contrôleur répond dans les temps. Néanmoins, ce n'est pas suffisant et il faut également vérifier que la réponse est sans anomalies. Pour cela, on propose de réestimer les variables internes du contrôleur et vérifier que les décisions sont vraisemblables vis-à-vis de ces estimations. Un premier observateur a été proposé sur un cas simple où le contrôleur est en charge de la commutation de niveau 2. On a pu se rendre compte que l'attaque ainsi que la défaillance ont bien été détectées. Par ailleurs, étant donné qu'on était dans un cas déterministe, on n'a pas eu de fausse alarme lors de la phase nominale. Pour construire cet observateur, nous avons utilisé la connaissance experte de l'algorithme de contrôle. Cette connaissance est utile, mais on verra qu'elle n'est pas forcément nécessaire. Dans la suite nous allons prendre du recul et limiter les hypothèses sur la connaissance de l'algorithme, en construisant l'observateur simplement à partir du type d'application considérée.

Dans le prochain chapitre, nous allons également développer la contribution scientifique visant à détecter des anomalies dans les réponses du contrôleur. Pour cela, des propriétés nécessaires sur ces réponses vont être introduites et enfin une analyse de la vraisemblance des décisions selon le type de contrôle, déterministe ou non, sera présentée.

Chapitre 3

Détection d'anomalies dans un contrôle de réseau : solutions originales

Nous venons de présenter le problème ainsi que le principe de la méthode que nous souhaitons mettre en place. Nous avons vu que nous allons aborder la détection d'intrusion à la fois par spécification et par détection d'anomalies. La spécification a été présentée, il s'agit d'établir le *template*, inspiré de (Pandalai & Holloway, 2000), du comportement de la commande. C'est-à-dire, on vérifie qu'à chaque requête des infrastructures, le contrôleur répond dans les temps. Cela correspond à la spécification du comportement. Maintenant, il faut vérifier que la réponse ne présente pas d'anomalies. Nous allons donc combiner, le premier modèle construit par spécification et un nouveau modèle que nous allons définir dans ce chapitre. Dans un premier temps, nous présenterons des conditions nécessaires à l'absence d'anomalies dans le contrôle. On verra qu'elles ne sont pas suffisantes et donc on proposera de les compléter, dans un second temps selon le type d'algorithme de contrôle, en fait selon qu'il soit déterministe ou non par une approche multicritère.

3.1 Conditions nécessaires à l'absence d'anomalie dans le contrôle

Dans cette partie nous allons établir des propriétés structurelles vis-à-vis des décisions prises par le contrôleur. Ces propriétés sont liées à l'application considérée et illustrées ici pour du routage. On va donc proposer en sus des propriétés temporelles, de nouvelles sur la structure du plan de données mis en place par le contrôleur.

3.1.1 Cohérence des routes

Pour rappel, le plan de données est l'ensemble des routes mises en place en réponse aux demandes sur le réseau. Ainsi, nous considérerons qu'un plan de données est cohérent (et donc sans anomalies) si l'ensemble des routes le sont. Pour cela nous proposons quatre propriétés à vérifier pour assurer la cohérence d'une route.

1. La route doit commencer depuis le switch à l'origine de la demande.
2. La route ne doit pas présenter de boucle. Dans le cas contraire, en présence de boucles, le réseau est alors sujet à des tempêtes de diffusion.
3. La route ne doit pas avoir de discontinuité dans le chemin (lorsque le contrôleur transmet à un switch la direction d'un second switch, il faut vérifier qu'il y a aussi une règle

pour le second switch). Sinon, le flux ne sera pas transmis et sera perdu au niveau de la discontinuité.

4. La route doit avoir une bonne terminaison : vérifier que la destination est au bout de la dernière instruction.

3.1.2 Formalisation

Formalisons ces règles. Comme vu, le plan de données à l'instant t reconstruit par l'observateur est défini par $\mathcal{P}(t)$ comme étant l'union de toutes les routes $\mu(t, s, d)$ pour lesquelles il y a une demande de la source s et en direction de d. On note $\mathcal{G} = (V, E)$: la topologie composée de l'ensemble des commutateurs, V et des liens entre les commutateurs, E et on introduit la fonction $\nu := i \rightarrow \omega$ qui renvoie l'identifiant du commutateur ω sur lequel un nœud i est connecté et $\xi := (\omega, \rho) \rightarrow \omega'$ la fonction qui renvoie le commutateur voisin ω' le long d'un port donné ρ d'un commutateur donné ω. Ainsi, chaque route de s à d incluse dans le plan au temps t sera considérée comme conforme à la topologie uniquement si les quatre règles suivantes sont vérifiées.

1. Le chemin commence au switch à l'origine de la demande :
   $\exists \sigma \in \mu\,(t, s, d) \mid \omega_\sigma = \nu\,(s)$
2. Pas de boucle :
   $\forall \sigma \in \mu\,(t, s, d)\,, \quad \nexists \sigma' \in \mu\,(t, s, d)\,,\ \sigma' \neq \sigma \mid \omega_\sigma = \omega_{\sigma'}$
3. Pas de discontinuité dans le chemin :
   $\forall \sigma \in \mu\,(t, s, d)\,, \quad \exists \sigma' \in \mu\,(t, s, d) \mid \xi\,(\omega_\sigma, \rho_\sigma) = \omega_{\sigma'}$
4. La destination est atteinte :
   $\exists \sigma \in \mu\,(t, s, d) \mid \omega_\sigma = \nu\,(d)$

Dans un cas d'application de type équilibrage de charge, on pourrait introduire des propriétés plus complexes sur le plan de données en analysant sa structure et la répartition des chemins par exemple.

Au final, notre méthode se représente comme en Fig. 3.1. L'observateur a vérifié que les commandes sont dans les temps, Fig, 2.8, et une fois qu'elles sont bien arrivées alors l'observateur vérifie qu'elles sont cohérentes vis-à-vis de l'application de contrôle, c'est-à-dire qu'elles vérifient les propriétés structurelles. Cependant, nous allons voir que ces propriétés bien que nécessaires présentent certaines limitations.

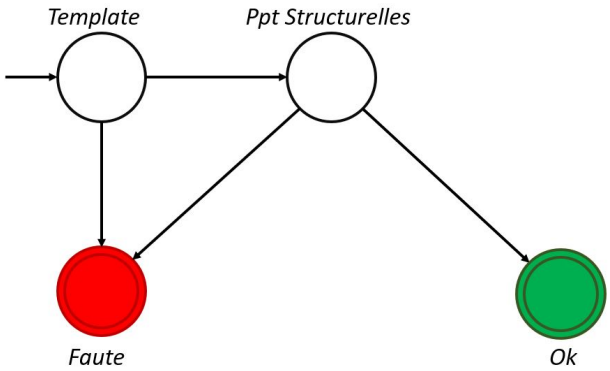


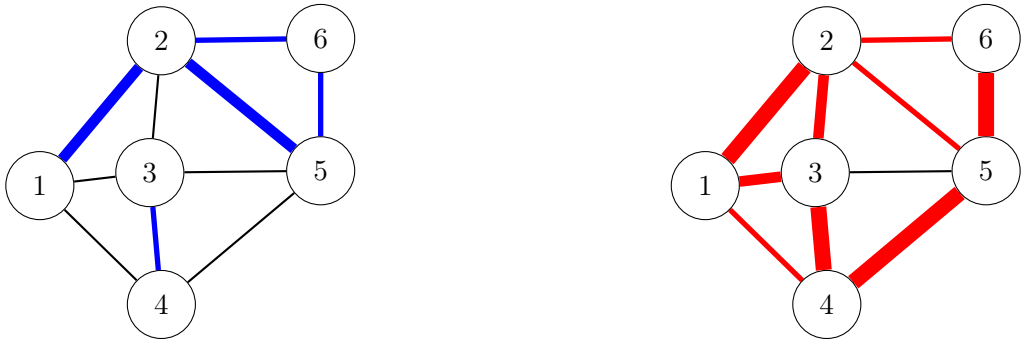
FIGURE 3.1 – Modèle de notre méthode de détection intégrant les propriétés structurelles.

3.1.3 Limites des propriétés structurelles

Considérons une topologie simple de 6 commutateurs comme donnée en Fig. 2.3 ainsi que 4 demandes à l'instant t tel que $\mathcal{D}(t) = \{(1,5),(1,6),(2,6),(3,4)\}$. Considérons les deux plans de données mis en place par le contrôleur suivant.

$$\Pi_N = \begin{pmatrix} 2 & 5 & 0 & 0 & 0 & 0 \\ 2 & 6 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 6 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \end{pmatrix} \quad \Pi_A = \begin{pmatrix} 2 & 3 & 4 & 5 & 0 & 0 \\ 4 & 5 & 2 & 3 & 6 & 0 \\ 3 & 1 & 4 & 5 & 6 & 0 \\ 2 & 6 & 1 & 0 & 4 & 5 \end{pmatrix}$$

Les chemins du plan de données Π_N sont représentés en Fig. 3.2a alors que ceux de Π_A sont représentés en Fig. 3.2b.



(a) Π_N (b) Π_A

FIGURE 3.2 – Plan de données mis en place suite à $\mathcal{D}$

Dans les deux cas, les plans de données sont cohérents vis-à-vis des propriétés définies ci-dessus. En effet, chacune des routes de Π_N et Π_A est cohérente. Cependant, il est clair que les routes de Π_A ne sont pas optimales dans la mesure où elles possèdent le nombre de sauts maximum (ou proche du maximum) possible sur le réseau. Ces routes sont alors assujetties à des latences importantes et conduisent à une qualité de service insatisfaisante.

Par conséquent, même si les conditions nécessaires définies ci-dessus sont satisfaites par le plan de données, il peut s'agir d'un plan de données malveillant visant à détériorer la qualité de service sur le réseau en mettant en place des routes qui augmentent la latence sur le réseau. Ces conditions ne sont donc pas suffisantes. Dans la suite, nous allons développer une approche afin de compléter ces conditions. Il y aura deux cas correspondant à deux hypothèses sur le contrôle : suivant qu'il soit déterministe (tel que Dijkstra) ou non déterministe (tel que des algorithmes génétiques).

En conclusion, en plus de vérifier que le contrôleur respecte la spécification établie et de vérifier que les commandes satisfont les propriétés structurelles définies selon l'application considérée, il est nécessaire d'analyser de surcroît la vraisemblance des décisions prises par le contrôleur (en rappelant que nous ne cherchons pas à les recalculer). Nous complétons donc notre méthode de détection comme représentée en Fig. 3.3. Nous commencerons par développer notre approche dans le cas déterministe.

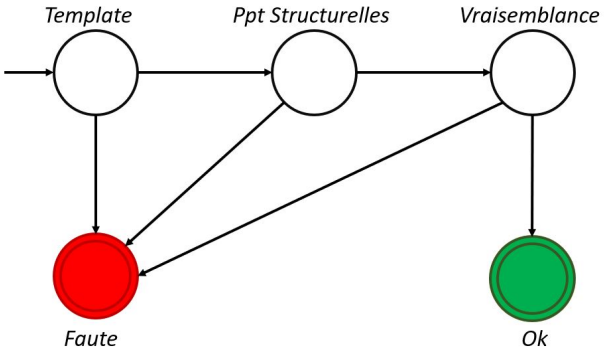


FIGURE 3.3 – Modèle de notre méthode de détection complet.

3.2 Cas d'un contrôle déterministe [Desgeorges et al., HPSR 2021]

L'observateur est censé déclarer une faute concernant les décisions du contrôleur si elles ne sont pas cohérentes par rapport aux contraintes définies dans la section précédente. Néanmoins il a été montré ce n'était pas suffisant et on va donc développer la méthode de détection dans le cas d'un algorithme de commande déterministe.

3.2.1 Construction de l'observateur

Ici, on suppose que l'algorithme de contrôle est déterministe, donc les décisions du contrôleur ne sont pas censées évoluer dans un contexte similaire. Ces décisions sont prises en fonction des variables internes du contrôleur et l'objectif de l'observateur va être de reconstruire la table de routage du contrôleur à partir des décisions qu'il prend. C'est-à-dire que lorsque le contrôleur mettra en place une route, cohérente, on supposera qu'il s'agit de la résultante de l'algorithme de commande. Puis, l'hypothèse de travail est que le contrôle est déterministe ce qui signifie que si le contrôle est sans anomalie, alors le contrôleur devra toujours prendre la même décision dans un contexte similaire.

Néanmoins, le contexte peut évoluer. Tout d'abord, une évolution de la topologie des infrastructures peut impacter le plan de données mis en place par le contrôleur. En reprenant l'exemple de Π_N, si le lien entre 1 et 2 se casse alors il est attendu que le contrôleur réagit afin de corriger le plan de données. Par ailleurs, selon les contrôleurs, il est possible qu'il ait à réagir en cas d'évolution de la demande (par exemple dans le cas où la transmission est terminée, certains algorithmes de contrôle suppriment le chemin en place).

La logique de l'observateur est donc de vérifier ces trois critères : les routes sont cohérentes, le contrôleur ne se contredit pas et il réagit de manière cohérente lorsque c'est nécessaire. Il y a donc une phase d'apprentissage nécessaire à l'enregistrement du chemin cohérent observé. Le chemin sera stocké dans l'ensemble $\mathcal{R}$.

3.2.2 Algorithme de détection

La logique de détection est présentée dans l'Algorithme 6. L'entrée est constituée des paquets OpenFlow correspondants à l'activité de la commande. À l'observation d'une requête, sous forme de `Packet_In`, qui n'a pas été vu précédemment alors la phase d'apprentissage commence à la ligne 3 : les mises à jour du plan de données sont stockées jusqu'à la fin du délai d'attente ou

que le chemin satisfait les propriétés introduites. Pour simplifier, nous introduisons une variable booléenne $cons(r)$ signifiant que si le chemin r satisfait les quatre contraintes, alors $cons(r) = vrai$; sinon, c'est $false$. Si la requête a été observée précédemment alors l'observateur vérifie que le contrôleur établit une route similaire que celle stockée dans $\mathcal{R}$ (ligne 15). Dans les deux cas, lorsque le chemin observé est supposé cohérent, il est ajouté à l'ensemble des chemins actuellement en place $\mathcal{R}_{Set}$. Cela permettra de déterminer les chemins impactés en cas d'évolution du contexte.

En effet, dès qu'une évolution de l'état de lien entre commutateurs est notifiée par un `Port_Status`, les chemins précédemment établis pourraient être impactés. Il y a deux possibilités : si le chemin est en place à l'observation du `Port_Status`, faisant partie de $\mathcal{R}_{Set}$, alors le contrôleur doit mettre à jour le chemin dès que possible en désinstallant les règles précédentes pour en installer de nouvelles. Cela implique que l'observateur doit recommencer une phase d'apprentissage, ligne 26, et laisser le contrôleur installer un nouveau chemin à partir du précédent. Les commandes peuvent être la suppression des règles précédentes, ligne 29, ou l'installation d'une nouvelle règle, ligne 31, ou la modification d'une règle précédente, ligne 33. À la fin du *timeout*, la cohérence de la route établie par les commandes observées est vérifiée (ligne 34). L'autre possibilité concerne les chemins appris, mais qui ne sont plus en service. Le contrôleur ne réagit pas directement au `Port_Status` puisqu'aucun chemin n'est impacté, mais à l'observation de la prochaine requête pour cette demande en établissant un nouveau chemin. L'observateur recommence donc la phase d'apprentissage, ce qui signifie que pour cette étape, il met à jour son estimation de la table de routage du contrôleur (ligne 40).

De plus, nous avons vu que les règles `Flow_Mod` ont une certaine durée de vie et finisse par expirer. Dans cette étude, le contrôleur supprime les chemins par suppression des règles précédentes (et ne fixe pas `idle_time`) lorsque le trafic demandé est terminé. A priori, le contrôleur ne reçoit pas les paquets liés au trafic donc ne sait pas lorsque le flux est terminé. En conséquence, le contrôleur demande périodiquement des statistiques, via des paquets `Multi_Part`, aux commutateurs. Ces statistiques indiquent le nombre d'octets qui ont été transmis et sont utilisées par le contrôleur afin de déterminer la continuité de la demande. S'il n'y a pas d'évolution, le contrôleur supprime le chemin en envoyant un paquet `Flow_Mod` pour supprimer les règles précédemment installées. L'observateur vérifie donc l'évolution du nombre d'octets transmis (ligne 37) et s'il n'y a pas d'évolution alors l'observateur attend la suppression du contrôleur et supprime ces chemins de $\mathcal{R}_{Set}$.

Enfin, il est nécessaire de vérifier que le contrôleur fonctionne correctement vis-à-vis de la spécification établie lors du chapitre précédent. Précisément, le contrôleur doit vérifier les propriétés temporelles. Ainsi, l'observateur vérifie que la tâche du contrôleur est effectuée à temps en surveillant le contrôleur par rapport au temps de tolérance *timeout* (lignes 5, 13, 27 et 43). Cette tolérance n'étant pas la même pour chaque tâche, puisque les algorithmes sont différents et donc la latence est différente, nous introduisons $timeout_{pin}$ (lignes 5 et 13), $timeout_{ps}$ (ligne 27) et $timeout_{mp}$ (ligne 43) qui correspondent au temps de réaction toléré dans le cas d'un paquet entrant, d'un état de port ou d'un `multipart`.

Algorithm 6: Logique de l'observateur dans le cas d'un algorithme de contrôle déterministe.

Input: Un paquet OpenFlow p.
Data: $\mathcal{R}$: l'ensemble des routes apprises et $\mathcal{G}$ le graphe de la topologie.

$p = wait(packet)$;
if $p \in \mathcal{I}$ **then**
  if $p \notin \mathcal{R}$ **then**
    $r = \emptyset$;
    while $\overline{cons(r)}\&timeout_{pin}$ **do**
      $f = wait(fmod)\&r = r \cup (p, fmod)$;
    if $cons(r)$ **then**
      $\mathcal{R} = \mathcal{R} \cup r$ & $\mathcal{R}_{Set} = \mathcal{R}_{Set} \cup r$;
    else
      return $Fault$;
  else
    $r_{learn} = r | r \in \mathcal{R}$ & $pin(r) = p$ & $r = r_{learn}$;
    while $path(r_{learn}) \neq \emptyset$ & $timeout_{pin}$ **do**
      $f = wait(fmod)$;
      if $f \notin path(r_{learn})$ **then**
        return $Fault$;
      else
        $path(r_{learn}) = path(r_{learn}) \setminus fmod$;
    if $path(r_{learn}) \neq \emptyset$ **then**
      return $Fault$;
    else
      $\mathcal{R}_{Set} = \mathcal{R}_{Set} \cup r$
else if $p \in \mathcal{N}$ **then**
  for $r_{learn} \in \mathcal{R} \mid \exists fmod \in cmd(r_{learn}) \mid port(fmod) = port(p)$ **do**
    $r'_{learn} = r_{learn}$;
    if $r_{learn} \in \mathcal{R}_{Set}$ **then**
      while $timeout_{ps}$ **do**
        $f = wait(fmod)$;
        if $type(f) = delete$ **then**
          $r'_{learn} = r'_{learn} \setminus f'$;
        else if $type(f) = add$ **then**
          $r'_{learn} = r'_{learn} \cup f$;
        else if $type(f) = modify$ **then**
          $r'_{learn} = r'_{learn} \setminus f' \& r'_{learn} = r'_{learn} \cup f$;
      if $\overline{cons(r'_{learn})} || r'_{learn} = r_{learn}$ **then**
        return $Fault$;
      else
        $\mathcal{R} = \mathcal{R} \setminus r_{learn} \& \mathcal{R} = \mathcal{R} \cup r'_{learn}$;
    else
      $\mathcal{R} = \mathcal{R} \setminus r_{learn}$
else if $p \in \mathcal{S}$ **then**
  if $byte_{t-2}(p) = byte_{t-1}(p) = byte(p)$ **then**
    while $timeout_{mp}$ **do**
      $f = wait(fmod)$;
      if $fmod \in \mathcal{R}_{Set}$ **then**
        $\mathcal{R}_{Set} = \mathcal{R}_{Set} \setminus fmod$
      else
        return $Fault$

3.2.3 Application

Dans cette section, l'algorithme de détection est mis en pratique et discuté selon la topologie représentée en Fig. 3.4. Le contrôleur est chargé du routage en utilisant l'algorithme Dijkstra et le trafic est appliqué est de l'hôte 10.0.1.1 (connecté au commutateur $of01$) vers l'hôte 10.0.1.2 (connecté au commutateur $of06$). Le *template* $Temp$ de l'algorithme est divisé en deux : à l'observation d'une requête, un plan de données (composé d'une seule route donc) est attendu et à l'observation d'une modification du statut d'un port, un nouveau plan de données est attendu pour corriger ce qui est impacté.

$$Temp = \{pin, \{((pout_i)_{i\in[1,n]}, it)\}\} \bigcup \{ps, \{((pout_i)_{i\in[1,m]}, it)\}\}$$

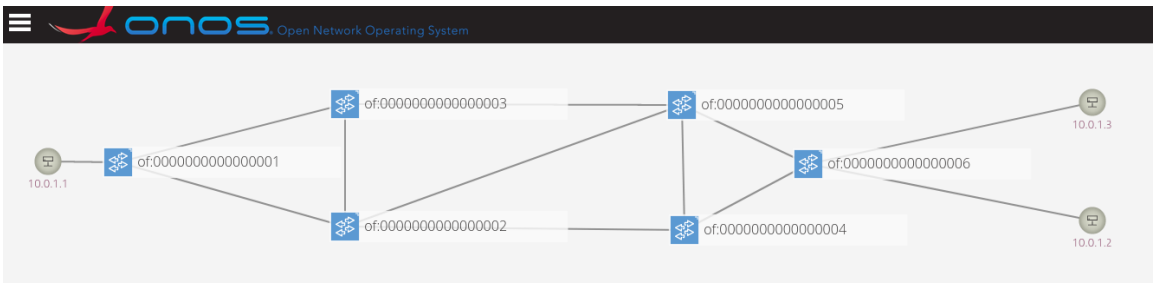


FIGURE 3.4 – Topologie considérée.

Dans les futures captures Wireshark l'identifiant d'un switch sera différent de celui en Fig. 3.4. Dans la Table. 3.1 nous donnons l'identifiant des commutateurs de Fig. 3.4 selon la trame Wireshark considérée. Par exemple, le commutateur identifié par $of01$ dans la Fig. 3.4, sera identifié par le numéro 5 dans la trame Wireshark donnée en Fig. 3.5.

Topologie	Fig. 3.5	Fig. 3.6	Fig. 3.7
$of01$	5	56	43
$of02$	4	58	46
$of03$	6	57	48
$of04$	2	54	47
$of05$	1	59	45
$of06$	3	55	44

TABLE 3.1 – Identifiants des commutateurs de la topologie de la Fig. 3.4

L'intervalle it est déterminé statistiquement comme présenté plus haut. Ici, on va déterminer la borne supérieure de ce temps d'attente en fonction de la tolérance qu'on se fixe vis-à-vis du pire cas observé. Pour simplifier la lecture, supposons que cette borne soit fixe (en réalité, elle évolue au fur et à mesure de l'expérience). Il y a deux intervalles différents, un pour une requête qui est un `Packet_In` it_{Pin} et un autre en réponse à un `Port_Status` it_{Ps}. Ici, on fixe $\beta = 1.5$ pour les deux. Le pire temps de réponse observé dans le cas d'un `Packet_In` est $6,34$ secondes alors que dans le cas d'un `Port_Status` on observe dans le pire cas un temps de réponse de $0,16$ seconde.

En conclusion les intervalles de temps sont les suivants : $it_{Pin} = [0, 9.51]$ et $it_{Ps} = [0, 0.24]$.

3.2.3.1 Phase d'apprentissage

Cette phase correspond à l'observation de requêtes sans *à priori* sur le contrôleur ($\mathcal{R} = \emptyset$). Les trames correspondantes, capturées à l'aide de Wireshark, sont représentées sur la Fig. 3.5.

Time	No.	Switch	Action	Type
3.1944…	464	5		OFPT_PACKET_IN
3.2048…	466	3	3	OFPT_PACKET_OUT
3.2059…	468	3		OFPT_PACKET_IN
3.2105…	469	5	1	OFPT_PACKET_OUT
10.387…	854	2	4,1	OFPT_FLOW_MOD
10.387…	855	3	3,2	OFPT_FLOW_MOD
10.387…	856	5	1,2	OFPT_FLOW_MOD
10.387…	857	4	2,1	OFPT_FLOW_MOD

FIGURE 3.5 – Paquets échangés lors de la phase d'apprentissage.

L'évolution de $\mathcal{R}$ selon Algo. 6 est présentée dans le tableau. 3.2. Pour simplifier le tableau, seul le chemin observé lié à la requête du paquet numéro 464 est considéré.

TABLE 3.2 – Évolution de $\mathcal{R}$

$$r_1 = \{pin_{464}, \{(pout_i)_{i\in[1,n]}, [0, 9.51]\}\}$$
$$r_2 = \{pin_{468}, \{(pout_i)_{i\in[1,n]}, [0, 9.51]\}\}$$

Paquet	$\mathcal{R}$	Chemin Observé
464	$\emptyset$	$\emptyset$
468	$\emptyset$	$\emptyset$
854	$\emptyset$	$\{fmod_{854}\}$
855	$\emptyset$	$\{fmod_{854}, fmod_{855}\}$
856	$\emptyset$	$\{fmod_{854}, fmod_{855}, fmod_{856}\}$
857	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$

Il y a l'observation de deux requêtes, le paquet 464 pour le flux depuis 10.0.1.1 et 468 pour le flux depuis 10.0.1.2 (notés pin_{464} et pin_{468}). L'observateur lance l'Algo. 6. La première action du contrôleur est de répondre à la requête ARP par un *Packet_ Out* (les trames numéro 466 et 469). Ces trames n'installent pas de chemin et ne sont donc pas considérées par Algo. 6 puisque seule la fonction de routage est observée.

Tout d'abord, le paquet numéro 854 est observé et est un $fmod$, noté $fmod_{854}$. Cette commande est divisée en deux actions, l'une concernant le pin_{464} et l'autre le pin_{468}. Elle est ajoutée au chemin actuel observé $r_{pin,464}$ et $r_{pin,468}$ par la ligne 7. Ensuite, la cohérence de ces chemins est vérifiée à la ligne 5. Ici, les chemins mis en place ne sont pas complets et ne respectent pas les propriétés introduites : il y a une discontinuité (puisque le chemin est pas encore complet) et les terminaisons ne sont pas atteintes. Ainsi, aucune décision n'est prise et l'observateur attend les autres commandes. Il en sera de même pour les paquets numéro 855, 856 et 857.

Maintenant, pour la trame numéro 857, notée $fmod_{857}$, le même processus est effectué sauf que les chemins ont satisfait les trois conditions introduites dans la section précédente, ce qui signifie que la condition $cons(r_{pin,464})$ est maintenant satisfaite. Le chemin est alors stocké dans $\mathcal{R}$. Nous avons supposé ici que la phase d'apprentissage n'était pas attaquée. Cependant, les

anomalies auraient également été détectées par une analyse de la cohérence des commandes lignes 8 et 10.

3.2.3.2 En production

Les trois cas suivants sont supposés apparaître après la phase d'apprentissage.

Cas d'une évolution de la topologie des infrastructures

Le premier cas considéré est l'évolution de la topologie de l'infrastructure. Ici, une panne du lien entre le switch 2 et 4 est considérée et notifiée par un *Port_ Status ps*. Les trames échangées dans ce cas, capturées avec Wireshark, sont données en Fig. 3.6.

Time	No.	Switch	Action	Command			Type	Port
134.990…	6574	54					OFPT_PORT_STATUS	s4-eth1
134.990…	6575	54					OFPT_PORT_STATUS	s4-eth1
134.991…	6576	58					OFPT_PORT_STATUS	s2-eth3
134.991…	6577	58					OFPT_PORT_STATUS	s2-eth3
135.038…	6580	59	4,1	OFPFC_ADD	OFPFC_ADD		OFPT_FLOW_MOD	
135.045…	6582	54		OFPFC_DELETE…	OFPFC_DELETE…		OFPT_FLOW_MOD	
135.047…	6587	55	3,2	OFPFC_DELETE…	OFPFC_ADD	OFPFC_MODIFY	OFPT_FLOW_MOD	
135.047…	6588	58	1,2	OFPFC_ADD	OFPFC_DELETE…	OFPFC_MODIFY	OFPT_FLOW_MOD	

FIGURE 3.6 – Paquets échangés suite à la notification de la défaillance d'un lien.

Le chemin établi pendant la phase d'apprentissage est impacté, ce qui signifie que l'observateur recommence une phase d'apprentissage. Ainsi, pendant les prochaines 240 ms, l'observateur attend un nouveau chemin. L'évolution du chemin établi est présentée dans le tableau. 3.3.

TABLE 3.3 – Évolution du chemin en place.

Paquet	Chemin Observé
6574	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$
6580	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}, fmod_{6577}\}$
6582	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{6577}\}$
6587	$\{fmod_{854}, fmod_{855}, fmod_{6587}, fmod_{6577}\}$
6588	$\{fmod_{854}, fmod_{6588}, fmod_{6587}, fmod_{6577}\}$

À la fin du *timeout*, la cohérence de la route finale $r_{Ps} = \{fmod_{854}, fmod_{6588}, fmod_{6587}, fmod_{6577}\}$ est testée. Ici, r_{Ps} est cohérent et a donc remplacé r_1 dans $\mathcal{R}$.

Cas d'attaque

L'attaque considérée est la même que précédemment, c'est-à-dire une modification du port de transmission par l'attaquant, que l'on peut modéliser par un biais b_{Cmd} d'une commande $fmod \in \mathcal{A}$ afin de donner la commande biaisée $fmod' \in \mathcal{A}$ tel que :

$\forall i \in [1, lengh(pout)] \quad pout'[i] = pout[i]$

$fmod'[1] = 10$ si $fmod[3] = of6$, $fmod'[1] = fmod[1]$ sinon

L'objectif d'un tel biais est de retransmettre tout le trafic qui passe par le commutateur $of06$ sur le port 10. L'objectif de ce qui suit est de montrer comment cette anomalie est détectée.

Après la phase d'apprentissage, à l'observation de chaque requête similaire, le même chemin est attendu. Les trames correspondantes, capturées à l'aide de Wireshark, sont données dans la Fig. 3.7.

Time	No.	Switch	Action	Type
67.983…	1569	43		OFPT_PACKET_IN
67.993…	1570	44	3	OFPT_PACKET_OUT
67.994…	1571	44		OFPT_PACKET_IN
67.996…	1573	43	1	OFPT_PACKET_OUT
71.189…	1759	46	1,2	OFPT_FLOW_MOD
71.192…	1762	43	2,1	OFPT_FLOW_MOD
71.194…	1766	47	4,1	OFPT_FLOW_MOD
71.195…	1769	44	10,10	OFPT_FLOW_MOD

FIGURE 3.7 – Paquets échangés dans la phase courante suite à une attaque.

L'évolution de $\mathcal{R}$ dans ce cas est donnée dans la Table. 3.4. Pour simplifier, on ne considère que le chemin attendu dans l'instance liée à la requête du paquet numéro 1569 (un `Packet_In`, noté pin_{1569}).

TABLE 3.4 – Évolution de $\mathcal{R}$ avec r_1 et r_2 les deux routes apprises lors de la phase d'apprentissage

Paquet	$\mathcal{R}$	Chemin attendu
pin_{1569}	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$
pin_{1571}	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}, fmod_{856}, fmod_{857}\}$
$fmod_{1759}$	$r_1 \cup r_2$	$\{, fmod_{854}, fmod_{855}, fmod_{856}\}$
$fmod_{1762}$	$r_1 \cup r_2$	$\{fmod_{854}, fmod_{855}\}$
$fmod_{1766}$	$r_1 \cup r_2$	$\{fmod_{855}\}$
$fmod_{1769}$	$r_1 \cup r_2$	$\{fmod_{855}\}$

Le `Packet_In` est observé à la trame numéro 1569, noté pin_{1569}, et cette requête a déjà été observée avec pin_{464} ce qui signifie que l'observateur a une estimation de la table de routage du contrôleur pour cette demande. En conséquence, le plan de données $r_1 \in \mathcal{R}$ est attendu. Pendant la phase d'exécution, l'objectif est de vérifier que le contrôleur prend des décisions similaires à celles de la phase d'apprentissage et donc qu'on retrouve le chemin r_1. Comme $fmod_{1759}$, $fmod_{1762}$ et $fmod_{1766}$ font partie du plan de données attendu, les commandes sont supposées cohérentes et sont donc supprimées du chemin attendu par la ligne 20 d'Algo. 6. En ce qui concerne $fmod_{1769}$, la commande ne fait partie d'aucun plan de données attendu et est donc supposée être incohérente. En conséquence, une erreur est déclarée ligne 18 de Algo. 6. Dans cet exemple, le biais considéré était une modification du port de transmission, mais tout autre biais de la commande, qui peut être formalisée comme présenté précédemment, serait détecté de manière similaire.

On peut constater que l'attaque a bien été détectée et que l'observateur a bien réagi aux réactions du contrôleur face à une requête et à une évolution de la topologie des infrastructures. Cependant, il est important de mentionner que bien qu'efficient, l'observateur pourra avoir une structure complexe et que son développement pourra nécessiter un effort non négligeable. L'observateur a été construit à partir du comportement supposé du contrôleur, mais ici nous avons

considéré que 3 tâches primitives : mise en place de chemin, réaction à une cassure de lien au niveau des commutateurs et suppressions des chemins lorsqu'ils ne sont pas nécessaires. Chacune de ses primitives correspond à un bloc de la structure de l'Algo 6. Sur un contrôleur en exploitation on peut imaginer plus de tâches comme la gestion des liens, la transmission de paquet Link Layer Discovery Protocol (LLDP) etc ... Auquel cas il faudrait compléter encore l'observateur en rajoutant de nouveaux blocs de gestion de ses primitives.

3.2.4 Limites vis à vis du cas non déterministe

Nous venons de proposer une méthode de détection d'anomalie dans le cadre d'un algorithme de commande déterministe. Si ces algorithmes ont prouvé leurs efficacités sur des petites architectures, il n'est pas toujours possible d'obtenir ainsi une solution optimale pour des problèmes plus complexes et ils ne passent pas à l'échelle d'où l'émergence de l'utilisation d'algorithme de Machine Learning ce contrôle de réseau, comme présenté dans (Hakiri, Berthou, Gokhale, & Abdellatif, 2015), (Boutaba et al., 2018) ou (Valadarsky, Schapira, Shahaf, & Tamar, 2017). En conséquence, il est possible d'observer plusieurs plans de données dans un même contexte. On a par exemple fait tourner l'algorithme de contrôle proposé par (Casas-Velasco et al., 2020) dans les mêmes conditions que leurs cas d'études (topologie du réseau GÉANT et trafic TOTEM [4] (également référencé par la librairie SNDlib [5])(TOTEM, January, 2006)) sur deux jours. La quantité d'information injectée est représentée en Fig. 3.8. Fig. 3.8b représente la quantité totale sur le réseau à chaque instant alors que Fig. 3.8a représente la quantité par demande.

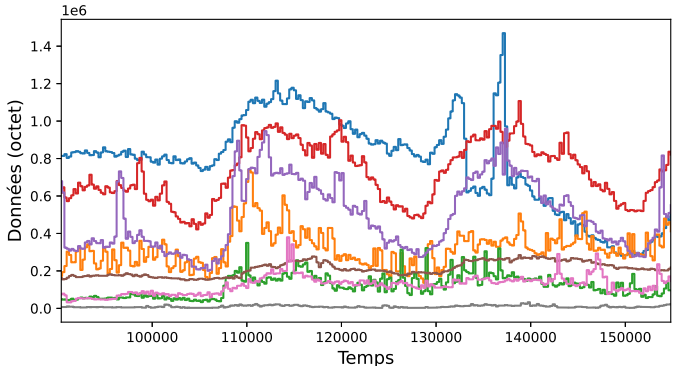
Le contrôleur doit mettre en place un nouveau plan de données toutes les 20 secondes selon une politique visant à minimiser des métriques en utilisant un algorithme de machine learning de type renforcement. Il y a 506 demandes et nous avons analysé la disparité des chemins mis en place par le contrôleur. Cette disparité est représentée en Fig. 3.9. Le premier graphe, Fig. 3.9a , représente le nombre de chemins différents observés pour chaque demande avec en abscisse la numérotation de la demande et en ordonnée le nombre de chemins différents pour cette demande. Cela permet de se rendre compte de la diversité des routes puisqu'il y a en moyenne 20 routes différentes par demande sur deux jours. Le second graphe, Fig. 3.9b , représente le nombre de chemins observés pour la première fois par plan de données. On peut voir que même après 2 jours il y a toujours des routes cohérentes et sans anomalies qu'on n'avait pas encore observées. Le troisième graphe, Fig. 3.9c , représente le nombre de chemins différents par rapport au plan précédent. Cela permet de voir qu'il est possible d'avoir une grande variance dans le contrôle puisque près de la moitié du plan de données est changé d'un plan à l'autre.

En conclusion, la méthode précédemment développée pour le cas d'un contrôle déterministe n'est donc plus applicable et les propriétés structurelles restent insuffisantes, donc une nouvelle méthode de détection est nécessaire.

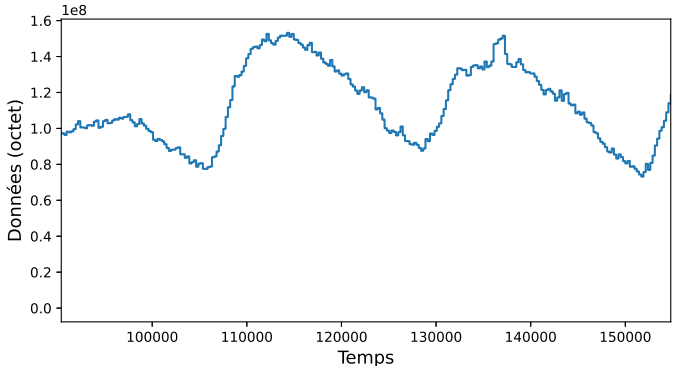
3.3 Cas d'un contrôle non déterministe [Desgeorges et al., TA 2022]

Précédemment, on vérifiait la cohérence du contrôle vis-à-vis des propriétés structurelles ou vis-à-vis de l'hypothèse de déterminisme. Ici, du fait du non-déterminisme, on ne peut plus appliquer cette méthode. On propose donc de déterminer le niveau de vraisemblance du plan mis en œuvre par le contrôleur. Par exemple, un plan dans lequel toutes les routes visitent

4. `https://totem.info.ucl.ac.be/dataset.html`
5. `http://sndlib.zib.de/home.action`



(a) Quantité émise par demandes.



(b) Quantité émise sur le réseau.

FIGURE 3.8 – Données injectées.

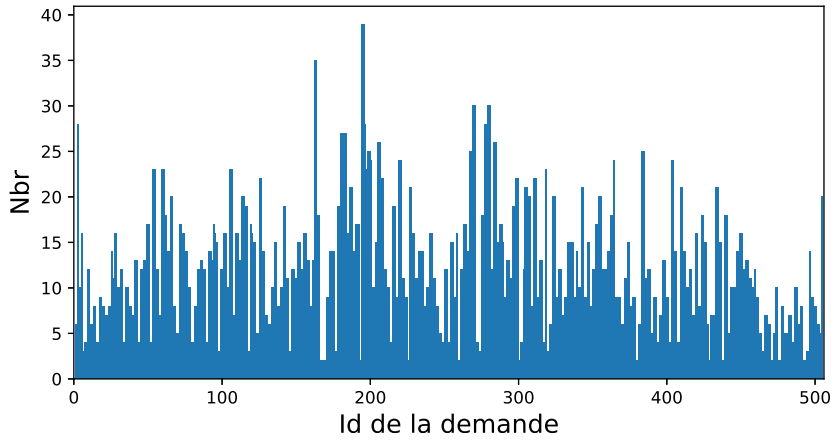
un commutateur commun pourrait être le symptôme d'un attaquant voulant provoquer une congestion dans le réseau et donc une augmentation de la latence de transmission des paquets. Déterminer si le plan est vraisemblable dépend des propriétés de l'algorithme de contrôle et nous allons proposer différentes formulations.

3.3.1 Construction de l'observateur

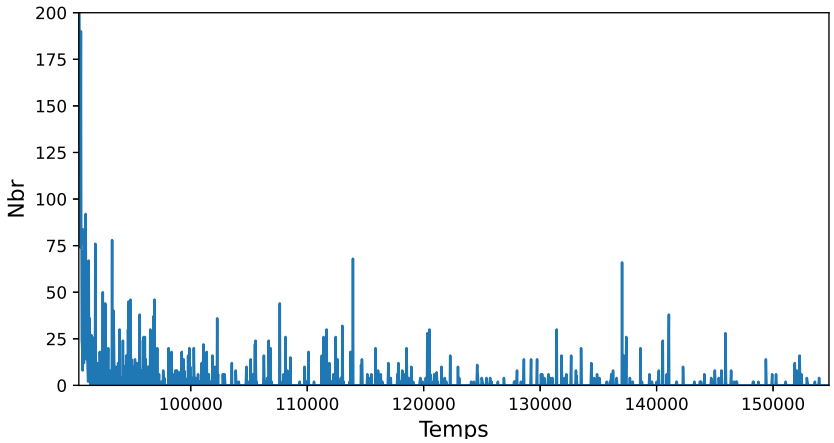
De la même façon que dans le cas déterministe, l'objectif est de détecter des anomalies à partir de la construction d'un modèle non fautif. Ainsi, le comportement observé est comparé aux modèles du comportement non piraté (observé lors d'une phase d'apprentissage). On se place dans le cas d'un algorithme de contrôle non déterministe, ce qui fait que notre évaluation d'une observation ne peut pas être binaire. Ainsi, comme on l'a vu dans la section. 2.3, nous allons établir un score de vraisemblance à chaque observation. Ainsi, par comparaisons et évaluations statistiques, nous proposons de déterminer le score de vraisemblance d'un plan de données $\mathscr{L}(\mathcal{P})$ selon une approche multicritères :

$$\mathscr{L}(\mathcal{P}) = \Sigma_{i=1}^{n} \alpha_i \times \hat{p}_i$$

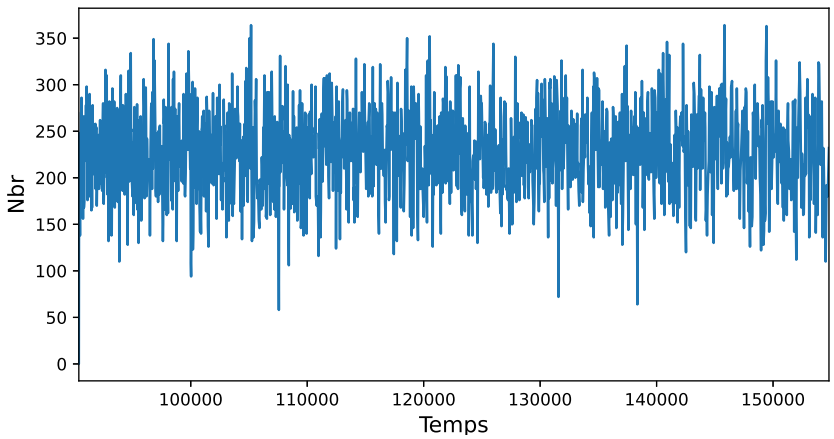
avec :
— $(\alpha_i)_{i\in[1,n]}$: le coefficient du critère i sachant que : $\Sigma_{i=1}^{n} \alpha_i = 1$
— $(\hat{p}_i)_{i\in[1,n]}$: le niveau de vraisemblance normalisé du critère i



(a) Histogramme du nombre de routes différentes observées par demande.



(b) Nombre de routes observées pour la première fois par plan.



(c) Nombre de routes différentes du plan précédent.

FIGURE 3.9 – Disparité dans le contrôle.

De plus, un plan de données $\mathcal{P}$ mis en œuvre par le contrôleur est supposé vraisemblable si son score de vraisemblance est supérieur à un seuil noté TD, c'est-à-dire si $\mathscr{L}(P) > TD$.

La vraisemblance de chaque critère p_i peut être obtenue à partir de plusieurs méthodes de calcul. Il est donc nécessaire de la normaliser et chaque valeur normalisée $\hat{p}_i$ est obtenue, avec P l'ensemble des valeurs utilisées lors de la phase d'apprentissage, comme suit :

$$\hat{p}_i = \frac{p_i - min(P)}{max(P) - min(P)} \tag{3.1}$$

Étant donné que P est déterminé lors de la phase d'apprentissage qui est nominal il est possible que des valeurs sous attaques dépassent $max(P)$ (ou soit inférieur de $min(P)$). Pour assurer que la valeur normalisée soit entre 0 et 1, on sature aux extrémités en retenant le min avec 1 (et le max avec 0).

Ces critères sont définis de manière générale. C'est-à-dire que leurs définitions formelles dépendent du cas d'application considéré. Ici, dans ce travail, en considérant toujours une application de routage, nous proposons les deux critères suivants :

— Critère 1 : évaluation de la performance des plans (p_1).

— Critère 2 : vraisemblance des routes (p_2).

En conséquence le niveau de vraisemblance est calculé comme suit :

$$\mathscr{L}(\mathcal{P}) = \alpha_1 \times \hat{p}_1 + \alpha_2 \times \hat{p}_2$$

Le processus de notre détection est représenté dans Figure 3.10. On y retrouve le processus de communication entre les commutateurs et le contrôleur avec d'un côté les requêtes, que ça soit $\mathcal{I}$, $\mathcal{P}$ ou $\mathcal{S}$, et de l'autre les commandes $\mathcal{A}$. Cette activité est observée par l'observateur, à travers les paquets OpenFlow, ce qui permet de reconstruire la demande et le plan de données en place. À partir de là, l'observateur va donc devoir établir un niveau de vraisemblance pour la commande observée O. En cas de grande variance dans le contrôle, il peut y avoir une étape de *clustering* afin de limiter le nombre d'observations. Cela va être détaillé, ainsi que la façon de déterminer le niveau de vraisemblance, ci-après.

Dans ce qui suit, nous commençons ainsi par développer le calcul de p_1 et p_2.

3.3.2 Critère 1 : évaluation des performances des plans

Le premier critère considéré concerne les performances des décisions prises par le contrôleur. En effet, le contrôleur a pour objectif de minimiser ou maximiser des métriques m au niveau des commutateurs (ça peut être le délai, la bande passante utilisée, etc ...). L'objectif de l'observateur est donc de vérifier que l'évolution de ces métriques est vraisemblable. L'idée étant que la Qualité de Service (QdS) ne doit pas significativement évoluer (hors modification topologique).

Parmi les indicateurs de QdS, nous nous intéresserons à la bande passante offerte par rapport au débit de la demande. Pour chaque flux (s, d), on compare le trafic envoyé N_{Sent} et celui reçu $N_{Received}$. Ces informations sont disponibles pour l'observateur grâce à l'observation des statistiques $\mathcal{S}$ envoyées par les commutateurs au contrôleur. Pour un flux donné (s, d), l'efficacité du plan est calculée comme suit :

$$m\left(\mathcal{P}, (s, d)\right) = \frac{N_{Sent} - N_{Received}}{N_{Sent}}$$

On ne peut rien déduire directement de la mesure absolue de cette valeur. En effet, l'impact peut diminuer (ce qui correspond à observer un décalage entre ce qui est reçu et ce qui est envoyé)

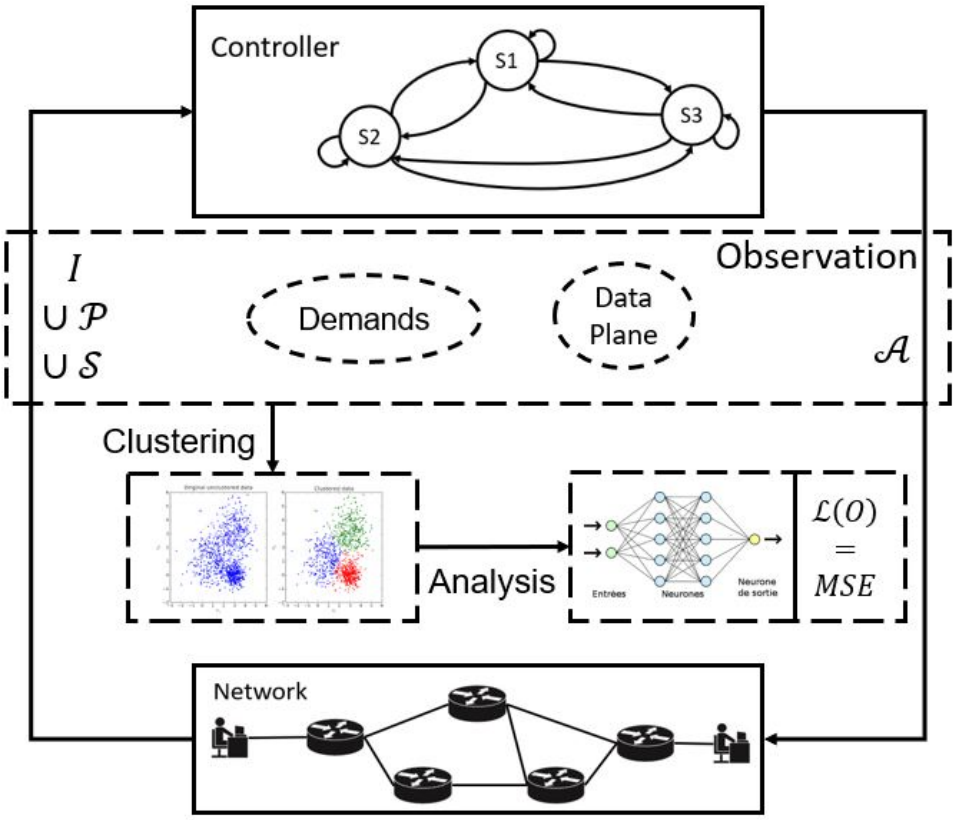


FIGURE 3.10 – Processus de détection générale

sans que ça soit de la faute du contrôle. Par exemple, dans le cas où une demande supérieure à la capacité des liens apparaît alors il va y avoir une latence dans la transmission des paquets, mais qui n'est pas due aux décisions du contrôleur. C'est la raison pour laquelle, on va se concentrer sur l'évolution de cette mesure plutôt que sur la valeur absolue à un instant. On propose pour cela d'estimer la dérivée de l'évolution de ces mesures en mesurant la distance entre la performance de deux plans et on introduit une fonction φ qui correspond à la distance entre deux plans $\mathcal{P}$ et $\mathcal{P}'$ (quelle que soit la demande) telle que :

$$\varphi \quad : \quad (\mathcal{P}, \mathcal{P}') \quad \mapsto \quad \max_{\forall (s,d) \in D(t)} |m\left(\mathcal{P}, (s,d)\right) - m\left(\mathcal{P}', (s,d)\right)|$$

Cette fonction mesure l'écart entre le plan courant $\mathcal{P}$ et un plan précédent observé $\mathcal{P}'$ en considérant l'écart maximal observé. Le choix de la fonction maximum ici est dans l'objectif de se concentrer sur le pire cas afin de conserver la plus grande distance possible (et donc maximiser la détection d'un écart, et donc d'une anomalie), mais il est à noter qu'il est possible de considérer d'autres opérateurs tels que le minimum ou la moyenne au lieu du maximum dans l'objectif d'être plus tolérant. Par ailleurs, on ne distingue pas les flux dans cette approche puisque cela n'apporte rien à la philosophie du critère, mais il est également envisageable d'isoler certains flux selon leurs niveaux de priorité.

Ainsi, la vraisemblance du n-ième plan de données, $\mathcal{P}_n$ correspond à la distance entre $\mathcal{P}_n$ et les *depth* derniers plans observés :

$$p_1 = 1 - \max_{i=n-depth-1 \ldots n-1} \varphi(\mathcal{P}_n, \mathcal{P}_i)$$

Le paramètre de la profondeur d'observation *depth* est important puisqu'on cherche l'impact maximum observé par rapport aux situations précédentes afin d'identifier si une diminution est liée à une évolution du contexte ou au contrôleur. Il est donc nécessaire de considérer des situations précédentes ayant lieu dans un contexte similaire afin que la comparaison ait un sens et le paramètre *depth* permet donc de définir le contexte de comparaison.

Une telle définition implique qu'un plan peut être considéré comme défectueux/attaqué tant qu'il est différent d'au moins un plan observé précédemment. De la même manière que pour la fonction φ, cette affirmation peut être légèrement modifiée en considérant une autre fonction comme le minimum ou la médiane.

Ce critère présente quelques faiblesses. En effet, dans le cas d'une attaque qui vise à dégrader lentement les performances, l'évolution entre les plans peut rester faible de telle sorte que le seuil TD ne sera pas atteint et la faute/attaque non détectée. C'est pourquoi le deuxième critère se concentrera sur la diversité des routes elles-mêmes.

3.3.3 Critère 2 : vraisemblance des routes

Le second critère consiste à analyser la structure des routes et des décisions du contrôleur. L'objectif est de déterminer le score de vraisemblance d'une séquence observée de décisions, i.e. une séquence de plan de données, du contrôle noté W.

$$p_2 = \mathscr{L}(W) \tag{3.2}$$

Cette vraisemblance $\mathscr{L}(W)$ peut être calculée par plusieurs méthodes et en utilisant plusieurs modèles. Cependant, les décisions du contrôleur dépendent de ses variables internes. Comme il n'y a pas d'interface est-ouest entre le contrôleur et l'observateur, nous n'avons pas accès à l'évolution de ces variables internes. La seule source d'information est l'activité du contrôleur (c.-à-d. les messages OpenFlow), résultant elle-même de l'évolution de ses variables internes. De la même façon que pour le cas déterministe, nous allons tenter d'estimer ces variables internes afin de déterminer le score de vraisemblance des plans de données mis en place selon trois approches. Ainsi, l'objectif du modèle choisi sera de prédire et d'établir un score de vraisemblance à chaque décision du contrôleur en se basant uniquement sur les décisions précédentes observées.

Il y a donc deux phases : une d'apprentissage et une autre d'exploitation. Lors de la phase d'apprentissage, un modèle non fautif de l'activité de la commande doit être établi. Il servira lors de la phase courante afin de mesurer l'écart entre l'activité de la commande observée et le modèle non fautif. Ces deux phases sont donc distinctes. Cependant, le contexte réseau évolue et donc pour que la comparaison entre le modèle non fautif et l'activité qu'on observe ait du sens, il est nécessaire de renforcer le modèle au fil de l'observation. On introduit donc une fenêtre glissante où le modèle se renforce au cours du temps à partir des observations vraisemblables lors de la phase courante.

Le choix de trois modèles pour l'apprentissage est fait de sorte à rajouter un niveau d'abstraction à chaque nouveau modèle. Tout d'abord, l'utilisation d'automate probabiliste classique visant à simplement étudier statistiquement le lien entre chaque plan de données puis les chaînes de Markov à état caché faisant quelque chose de similaire, mais en rajoutant un niveau supérieur correspondant à des états cachés (en lien avec les états internes du contrôleur). Les chaînes de Markov à états cachés correspondent à une approche discrète et donc le troisième modèle sera une approche similaire, mais continue : les réseaux de neurones récurrents. Nous avons pu voir dans le chapitre 2 l'utilité de ces modèles pour la détection d'anomalies et nous avons considéré également PFA pour voir si ce niveau d'abstraction est nécessaire.

3.3.4 Solutions proposées pour l'apprentissage

Comme nous considérons un contrôle non déterministe, un événement non habituel peut se produire sans être un problème ou à cause d'un contrôleur malveillant. Il est donc important de considérer une séquence de décision plutôt qu'une seule décision isolée. Par conséquent, nous proposons de classer et de déterminer la vraisemblance d'une séquence d'n observations. Dans cet objectif, nous proposons d'utiliser trois modèles : Réseau neuronal récurrent (RNN), modèle de Markov caché (HMM) et Automate fini probabiliste (PFA) qui sont présentés ci-après.

Il est important de noter que d'autres modèles auraient également pu être utilisés. Le choix ici a été fait afin de voir les bénéfices de chaque niveau d'abstraction dans la construction du modèle de l'activité de la commande selon le type de commande. En effet, dans le chapitre suivant, on étudiera le comportement de chacun de ces modèles selon la variance du contrôle. Mais, en aucun cas on ne cherche à obtenir un modèle optimal puisque cette optimalité est dépendante du contrôle et donc on ne prétend pas présenter les modèles les plus performants .

L'idée des modèles est d'inférer, à partir des observations, sur les variables internes du contrôleur. Ici, comme il n'y a pas d'interface est-ouest avec le contrôleur, nous n'avons pas accès à l'évolution des variables internes de l'algorithme de commande. Nous n'avons accès qu'à l'observation de l'activité de la commande qui est l'événement résultant de l'évolution des variables internes et nous proposons comme seconde approche d'inférer sur les variables internes. Le processus est représenté dans la Fig. 3.11.

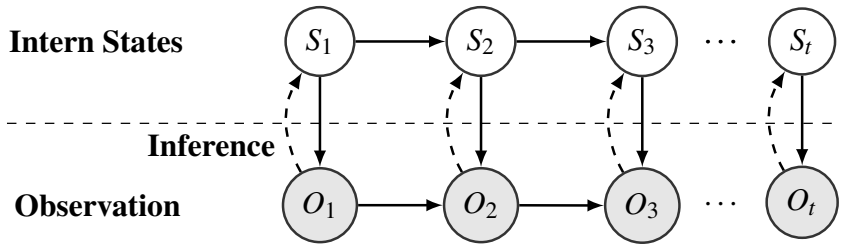


FIGURE 3.11 – Processus d'inférence

3.3.4.1 Chaînes de Markov à États Cachés

Nous supposons que l'évolution suit un processus de Markov. En effet, les décisions du contrôleur ne dépendent que de l'état du réseau au même instant qui est lié uniquement à la décision précédente. Ainsi, nous proposons d'utiliser le modèle du modèle de Markov caché (HMM). La définition générale du HMM englobe le cas où les variables cachées sont continues (voir (Cappé, Moulines, & Rydén, 2009)), mais ici nous considérerons la définition la plus répandue et la plus courante : les variables discrètes (Rabiner, 1989). Ce modèle a déjà été utilisé dans le contexte de la sécurité du contrôleur SDN comme dans (W. Wang et al., 2018) qui utilisent le HMM pour détecter des attaques de type DDoS. Ils profitent de la centralisation du contrôleur SDN pour collecter les informations des flux afin de vérifier que les flux sont vraisemblables ou non. Le principe de ce modèle est de déduire l'évolution des variables internes du contrôleur à partir de l'observation de la résultante de ces variables, et ensuite de déterminer la vraisemblance d'une observation par inférence sur les états internes. Dans cet objectif, la théorie probabiliste telle que le théorème de Bayes est utilisée.

Définition d'HMM

Un modèle de Markov caché (HMM) est défini par un triplet $HMM = (\pi, A, B)$ défini comme suit :

- N : le nombre d'états.
- $S = \{s_1, \dots s_N\}$: l'ensemble des états.
- M : le nombre d'observations.
- $O = \{o_1, \dots o_N\}$: l'ensemble des observations.
- $A \in M_{N,N}$: la matrice de transition (entre les états),
  $A = (a_{i,j} = p(q_t = s_j | q_{t-1} = s_i)$: représente la probabilité de passer de l'état s_i à l'état s_j. $\forall i : \Sigma_{j=1}^{N} a_{i,j} = 1$
- $B \in M_{N,M}$: la matrice de probabilité d'observation,
  $B = (b_{i,j} = p(ot_t = o_i | st_t = s_j))$: représente la probabilité d'observer o_i depuis l'état s_j. $\forall i : \Sigma_{j=1}^{N} b_{i,j} = 1$
- $\pi \in M_{1,N}$: distribution initiale des états.

Une représentation d'un HMM est donnée en Fig. 3.12.

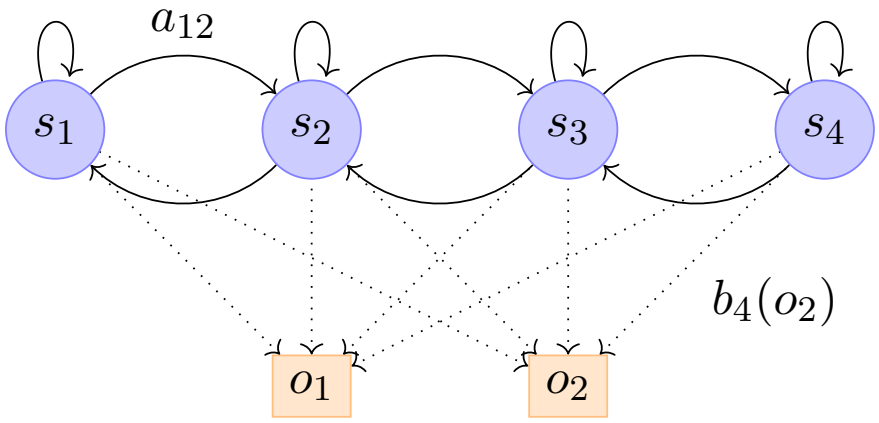


FIGURE 3.12 – Exemple d'une chaîne de Markov à état caché

Ici, les observations sont les plans de données mis en place par le contrôleur tandis que les états internes représentant symboliquement ceux du contrôleur. L'objectif va être de retracer l'évolution des variables internes du contrôleur ayant donné lieu à la séquence de plan de données observée.

Vraisemblance en utilisant un HMM

Formellement, HMM a introduit les trois problèmes développés dans (Rabiner, 1989) et est résumé comme suit :

1. Étant donné un HMM $\lambda = (\pi, A, B)$ et une séquence d'observation O, déterminer la vraisemblance $p_2 = P(O|\lambda)$.
2. Étant donné une séquence d'observation O et un HMM $\lambda = (\pi, A, B)$, déterminer la séquence d'état caché la plus probable Q ayant émis O.
   $Q^* = argmax_Q p(Q|O, \lambda)$
3. Étant donné une séquence d'observation O, un ensemble d'états S, déterminer les paramètres du HMM que sont π, A and B.
   $\lambda^* = argmax_\lambda p(O|\lambda)$

Pour résoudre chacun de ses problèmes, il y a dans la littérature des algorithmes bien connus comme :

1. Algorithme de Forward-Backward (Rabiner, 1989)
2. Algorithme de Viterbi (Forney, 1973) (Viterbi, 1971)
3. Algorithme de Baum-Welch (Baum, Petrie, Soules, & Weiss, 1970)

Ainsi, pour déterminer la vraisemblance d'une séquence d'observation, on peut utiliser l'algorithme Forward, tandis que l'algorithme de Baum-Welch est utilisé pour la phase d'apprentissage.

Cependant, les HMM ont une limitation fondamentale. L'algorithme de Viterbi, utilisé pour déterminer la meilleure séquence d'états cachés compte tenu d'une observation, est coûteux, tant en termes de mémoire que de temps de calcul. Pour une séquence de longueur n, la programmation dynamique pour trouver le meilleur chemin dans un modèle à N états prend un temps proportionnel à $N^2 \times n$. D'autres algorithmes pour les modèles de Markov cachés, comme l'algorithme forward-backward, sont encore plus coûteux et prennent un temps proportionnel à N^n.

3.3.4.2 Réseaux de Neurones Récurrents

Les réseaux de neurones sont de plus en plus populaires pour la détection d'anomalies (Kwon et al., 2019). Le deep learning a fait l'objet d'une grande attention au cours des dernières années, et de nouvelles techniques d'apprentissage profond apparaissent avec des fonctionnalités améliorées. De nombreuses applications informatiques et de réseau utilisent activement ces algorithmes d'apprentissage profond et font état de performances améliorées grâce à eux (Zhang, 2000). Il existe différents modèles associés aux réseaux de neurones comme par exemple Convolutional Neural Network ou Multi-layer Perceptron par exemple (Sarker, 2021). Bien qu'ils peuvent être utilisés, l'objectif de cette proposition n'est pas de faire un comparatif entre ces algorithmes puisque ce ne serait qu'utile dans le but de déterminer une solution optimale. Or, cette solution est dépendante du cas d'étude. Ainsi, on va en choisir un et ce sont les Réseaux de Neurones Récurrents (RNN) pour leurs proximités avec HMM.

Les RNN et les HMM partagent des similitudes puisqu'ils impliquent tous deux des variables latentes, mais ils diffèrent quant à la manière dont ces variables sont construites (Salaün, Petetin, & Desbouvries, 2019). La principale différence entre ces deux modèles est que les HMM constituent une approche discrète du problème alors que le RNN est une approche continue.

Définition d'un RNN

Un RNN peut être considéré comme une extension d'un réseau neuronal (NN) à action directe, où la sortie d'un état caché h_t dépend du temps précédent h_{t-1} et est calculée de manière séquentielle et déterministe par une fonction d'activation donnée σ qui dépend d'un ensemble de paramètres $\theta = (W, U, b_h)$ avec :

— x_t : vecteur d'entrée
— h_t : vecteur de la couche cachée
— W : paramètre matrice de connexion entre la couche cachée
— U : paramètre matrice de connexion entre la couche d'entrée et les couches cachées
— b_h : vecteur de paramètres
— σ : fonction d'activation

Et ensuite :

$$h_t = \sigma(W \times x_t + U \times h_{t-1} + b_h), \text{ for } t \in [1, T] \tag{3.3}$$

Un exemple de RNN est donné en Fig. 3.13

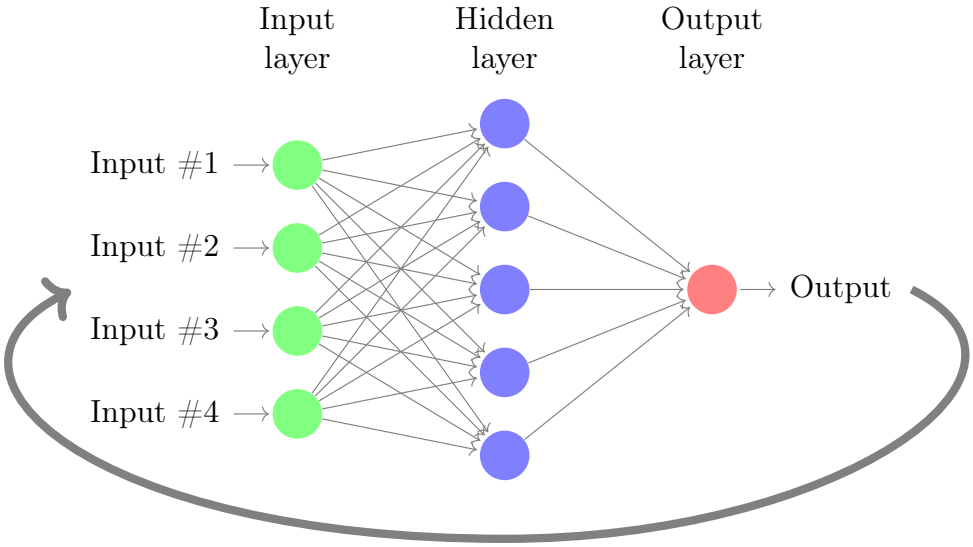


FIGURE 3.13 – Représentation d'un RNN

Vraisemblance en utilisant un RNN

Pour mesurer la vraisemblance d'une séquence dans un RNN, nous proposons de mesurer l'écart entre la prédiction comme dans (J. Liu et al., 2019) ou (Kolbæk, Yu, Tan, & Jensen, 2017), qui correspond au comportement attendu, $(\hat{O}_i)_{i\in[1,n]}$ et la séquence observée $(O_i)_{i\in[1,n]}$ en déterminant les erreurs quadratiques moyennes MSE. En conséquence :

$$p_2 = 1 - MSE = 1 - \frac{1}{n} \times \Sigma_{i=1}^{n}(O_i - \hat{O}_i)^2 \tag{3.4}$$

Enfin, nous venons de présenter deux solutions d'apprentissages classiques qui utilisent donc une couche d'abstraction représentant finalement une boîte noire. Pour voir l'utilité de cette couche, nous allons utiliser un troisième modèle.

3.3.4.3 Automate Fini Probabiliste

Une première approche consiste à considérer uniquement la séquence de plan de données sans tenir compte des états internes du contrôleur.

Définition de PFA

Dans cet objectif, nous avons proposé d'utiliser un automate fini probabiliste (PFA) (Bertrand et al., 2014) défini par un 5-tuple :

$$PFA = \langle Q_A, q_0, \Sigma_A, \delta_A, P \rangle \tag{3.5}$$

avec :

— Q_A est un ensemble non vide d'états.
— $q_0 \in Q_A$ est l'état initial.
— Σ_A est l'alphabet.
— $\delta_A = Q_A \times \Sigma_A \times Q_A$ est l'ensemble des transitions composé d'un état de départ, un événement conditionnant le franchissement de la transition et l'état d'arrivée.
— $P : \delta_A \to [0, 1]$ est la probabilité de transition.

Il convient de mentionner que pour chaque état, la somme de toutes les transitions sortantes est égale à 1 :

$$\forall q \in Q_A : \Sigma_{q' \in Q_A, o \in \Sigma_A} P((q, o, q')) = 1 \tag{3.6}$$

Un exemple de PFA est donné en Fig. 3.14.

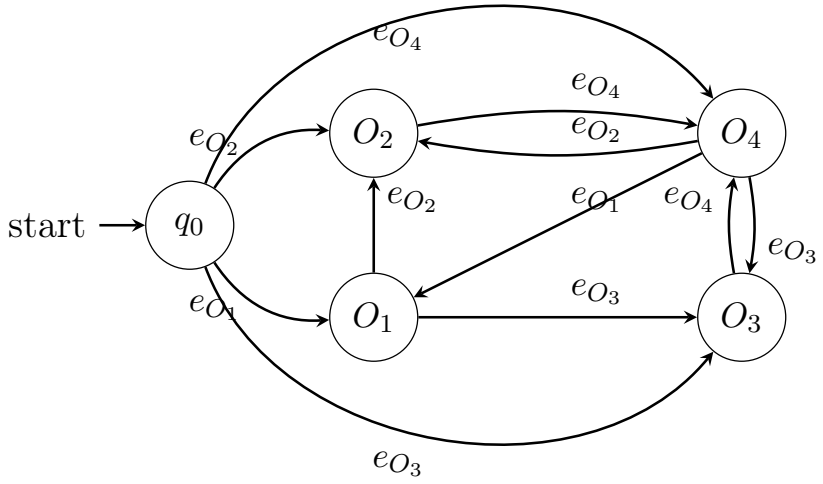


FIGURE 3.14 – Exemple d'un PFA

Il existe plusieurs façons de construire le PFA d'un système par connaissance experte ou bien par apprentissage. Ici, nous allons nous inspirer des travaux de (Fouquet et al., 2020) afin de construire le PFA. Dans notre problème les états correspondent à un plan de données et les événements correspondent à la mise en place d'un nouveau plan de données et donc la transition $\delta_{ex} = (\Pi_1, e_{\Pi_2}, \Pi_2)$ signifie que depuis le plan Π_1, le contrôleur a mis en place un nouveau plan de données : Π_2. Les probabilités de transition permettent d'évaluer la probabilité de voir un tel changement. Par exemple, si $P(\delta_{ex}) = 0,99$ alors lorsque Π_1 est en place il y a 99% de chance de voir Π_2 être mis en place. Ces probabilités sont déterminées statistiquement. À partir de ce modèle, nous allons évaluer la vraisemblance d'une séquence de décisions du contrôleur.

Vraisemblance en utilisant un PFA

Dans ce travail, nous considérons l'activité d'un contrôle non déterministe. Ainsi, l'apparition d'un événement inhabituel n'est pas un problème dans la mesure où l'on suppose que cela est possible. Dans cette perspective, nous considérerons la vraisemblance de la séquence d'événements n.

Ainsi, une trace correspondant à l'observation de la séquence de n plans $\mathcal{T} = (\mathcal{P}_i)_{i \in [1,n]}$ implique le franchissement de $(\delta_i)_{i \in [1,n]}$ dans le PFA θ. En conséquence :

$$p_2 = p_\theta(\mathcal{T}) = \Pi_{i=1}^{n} \frac{P(\delta_i)}{n} \tag{3.7}$$

Cependant, cette approche est limitée et n'est pertinente que pour un contrôle dont la variance est limitée. En effet, dans le cadre ou la variance augmente les décisions tendant vers une loi uniforme et donc l'écart entre les probabilités du modèle se nivelle. Il est alors nécessaire d'augmenter la profondeur de la séquence pour distinguer les niveaux de vraisemblance des séquences. Or, bien que théoriquement il n'y ait pas de problèmes à augmenter la profondeur de séquence, il faut retourner au problème applicatif. Cette augmentation de profondeur implique un élargissement du contexte considéré ce qui n'est pas toujours possible (apparition de demande, lien cassé au niveau des infrastructures).

3.3.4.4 Étape de regroupement pour réduire le nombre d'observations

Selon les contextes le nombre de plans de données possibles est trop important pour pouvoir être géré par les trois modèles proposés. Par exemple, si nous considérons le contrôle du routage du réseau GÉANT avec 23 nœuds, il y a plus de 2^{506} plans de données possibles. Ce minorant est calculé dans le cas ou seulement deux chemins par demande sont possibles. Or, dans un réseau réel, il y a de la redondance de chemin et selon la variance de la commande, on peut s'attendre à un grand nombre d'états. L'objectif est de rassembler les décisions qui peuvent être considérées comme similaires en fonction d'un ensemble de caractéristiques prédéfinies. Un exemple de caractéristiques dans le cas d'un contrôle de routage pourrait être le nombre total de sauts dans le plan de données. Par conséquent, chaque plan de données qui a un nombre similaire de sauts serait considéré comme équivalent et serait rassemblé dans un seul et unique état. Enfin, chaque état d'observation correspond à un cluster.

Pour déterminer les clusters il y a plusieurs algorithmes et façons de faire (Madhulatha, 2012) comme par exemple les méthodes hiérarchiques qui viseront à regrouper les éléments qui sont les plus proches entre eux selon une définition de la distance prédéfinie (la distance Euclidienne par exemple). Il y a ensuite les méthodes par partition qui sont basées sur la spécification d'un nombre initial de groupes, et la réaffectation itérative des objets entre les groupes jusqu'à la convergence. Un tel algorithme détermine généralement tous les groupes en une seule fois. La plupart des applications adoptent l'une des heuristiques populaires qu'est le k-Means (Likas, Vlassis, & Verbeek, 2003). Et enfin, il y a les méthodes par densité qui sont conçues pour découvrir des *clusters* de forme arbitraire. Dans cette approche, un *cluster* est considéré comme une région dans laquelle la densité des objets de données dépasse un seuil.

Nous ne détaillerons pas plus puisque cela ne sera pas appliqué dans notre cas d'étude, mais nous mentionnons la possibilité face à une trop grande diversité dans le contrôle. Il est également important de considérer qu'une telle approche a des limites. La plupart des algorithmes demandent en amont de définir le nombre de partitions. Cela demande d'en avoir une idée précise et même s'il existe des méthodes (comme par exemple la méthode d'Elbow (Bholowalia & Kumar, 2014)) pour optimiser sur certains jeux de données, mais elle n'est pas infaillible. Un mauvais regroupement des observations aura pour conséquence un mauvais apprentissage et donc la construction d'un mauvais modèle.

3.3.5 Preuve de concept

3.3.5.1 Environnement d'étude

À titre d'étude de cas, l'observateur proposé est appliqué au contrôleur de réseau présenté dans (Casas-Velasco et al., 2020). Ici, la fonction de contrôle est un routage proactif multiobjectifs par une technique d'apprentissage par renforcement. Les métriques utilisées sont le retard, la perte de paquets et la bande passante disponible. Le comportement du contrôleur est également proactif. Ainsi, périodiquement, un nouveau plan de données est mis en place par le contrôleur, quels que soient la demande et le trafic. Le code est disponible sur (Casas-Velasco, 2020) et est implémenté par un contrôleur Ryu (Team, 2012). La topologie se compose de 23 nœuds et de 37 liens comme indiqué dans la Fig. 3.15. Pour faire face aux limitations de Mininet, nous avons mis à l'échelle les capacités de liaison de 10 Gb/s, 2,5 Gb/s et 155 Mb/s de GÉANT à respectivement 100 Mb/s, 25 Mb/s et 1,55 Mb/s.

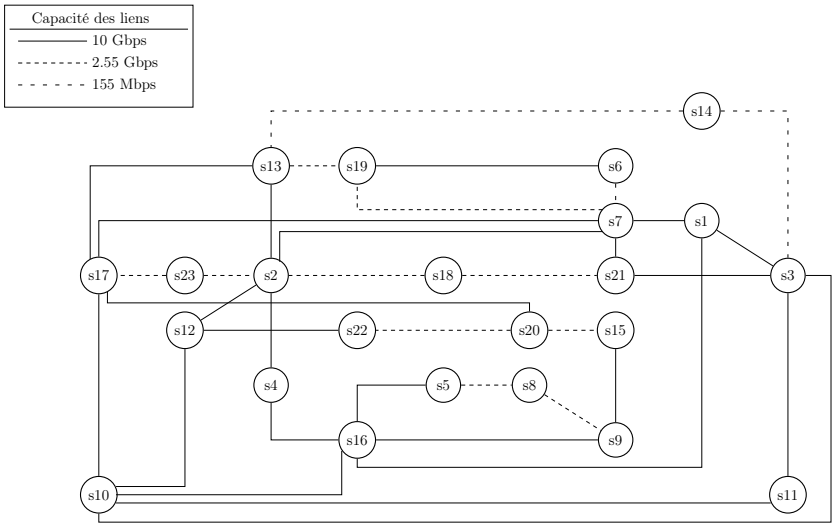


FIGURE 3.15 – Topologie du réseau GÉANT à 23 nœuds.

Le trafic considéré est le jeu de données TOTEM [6] (également référencé par la librairie SNDlib [7]) qui fournit des matrices de trafic intradomaines pour la topologie GÉANT. L'attaque considérée correspond à une modification de `Flow_Mod` telle que présentée dans (Lee et al., 2017b). Premièrement, l'attaquant prend le contrôle du contrôleur en utilisant Metasploit [8], une bibliothèque de Kali Linux. L'attaque consiste à utiliser une faiblesse sur la machine du contrôleur (ici, *vsftpd 2.3.4 backdoor exploitation*) pour y ouvrir une session et ensuite mettre en place un plan de données malveillant dont l'objectif est de rediriger le trafic vers le lien ayant la capacité minimale pour provoquer une saturation (et donc la latence sur le réseau). Deuxièmement, le plan de données calculé par le contrôleur est modifié afin de rediriger le trafic vers le lien ayant le moins de capacité dans le but de provoquer une congestion et par conséquent de dégrader le service (sans pour autant émettre un trafic additionnel sur le réseau).

La topologie de simulation est représentée en Fig. 3.16.

6. `https://totem.info.ucl.ac.be/dataset.html`
7. `http://sndlib.zib.de/home.action`
8. https ://www.metasploit.com/

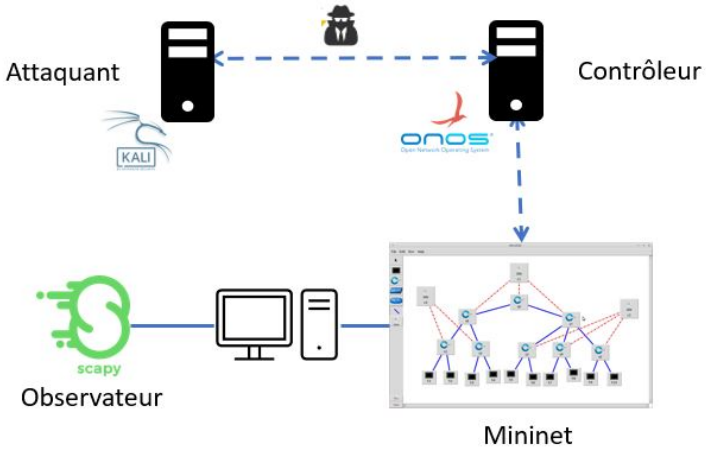


FIGURE 3.16 – Topologie physique

3.3.5.2 Impact des critères de vraisemblance

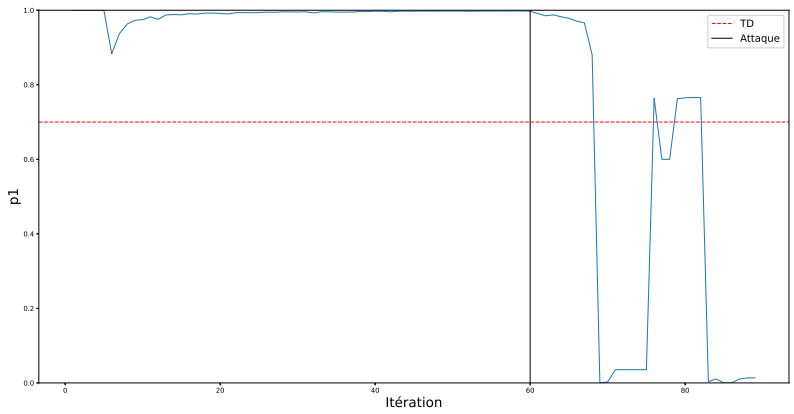
Avant de continuer, on rappelle que les valeurs des critères sont normalisées de sorte qu'elles puissent être additionnées pour obtenir le score de vraisemblance.

Dans cette partie, le résultat de la détection est étudié. Le contrôleur fonctionne avec l'algorithme de contrôle proposé dans (Casas-Velasco et al., 2020). Aussi, dans cette partie nous considérerons $\alpha_1 = 0,5$, $\alpha_1 = 0,9$ et $\alpha_1 = 0,1$ (donc respectivement $\alpha_2 = 0,5$, $\alpha_2 = 0,1$ et $\alpha_2 = 0,9$). L'expérience a été mise en place avec les trois modèles présentés précédemment et les conclusions sont les mêmes donc pour ne pas alourdir la lecture, nous n'en utilisons qu'un seul, le PFA ici. Les résultats des deux critères sont représentés sur la Fig. 3.17 et le score de vraisemblance est lui représenté en Fig. 3.18. Le premier graphique représente la vraisemblance calculée pour les trois valeurs de α_1, le deuxième graphique représente l'évolution normalisée du critère lié à la performance des décisions (p_1). Enfin, Fig. 3.18 représente l'évolution normalisée du score lié à la structure des décisions (p_2). Le déroulement est le suivant : 40 minutes de comportement nominal (jusqu'à l'itération numéro 60 sur la Fig. 3.17) et 20 minutes d'attaque (à partir de l'itération numéro 60 et jusqu'à l'itération numéro 89 sur la Fig. 3.17). Le seuil TD a été déterminé expérimentalement en fonction du pire cas observé et fixé à $0,7$.

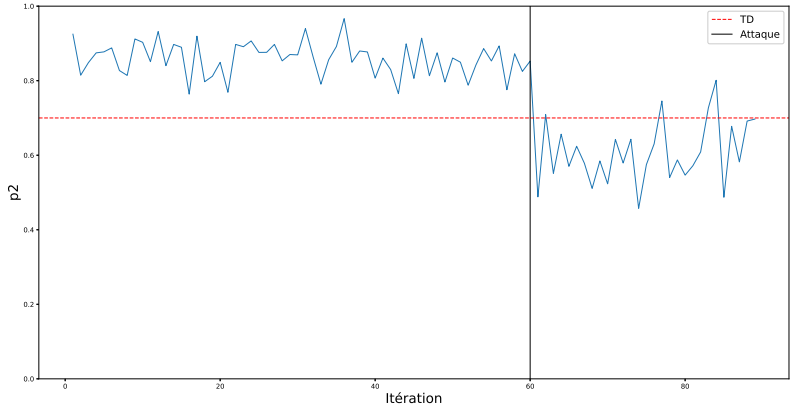
On peut noter que l'observateur a levé des alarmes et a donc bien détecté l'attaque. On peut également noter la stabilité des deux critères lors de la phase nominale et l'absence d'alarme (même si on ne peut pas généraliser comme on le verra dans le chapitre suivant).

Les alarmes soulevées pendant les expériences sont représentées par un point sur la Fig. 3.18. Il y a plusieurs faux négatifs après les attaques pour $\alpha_1 = 0,5$ et $\alpha_1 = 0,9$ (par exemple de la 61ème à la 65ème itération). Ces faux négatifs sont dus au fait que le critère 1 soit priorisé alors qu'on peut constater sur Fig. 3.18 que l'attaque n'a quasiment pas d'impact à ce moment-là sur ce critère contrairement au critère 2 comme on peut le voir sur la Fig. 3.17b. Ainsi, l'impact sur p_1 n'est toujours pas significatif et donc la moyenne avec p_2 reste au-dessus du seuil même si une diminution est observée sur la Fig. 3.17a et Fig. 3.18. Par conséquent, une priorisation de p_2 aurait permis de détecter l'attaque plus tôt dans ce cas comme on peut l'observer pour $\alpha_1 = 0,1$.

Néanmoins, le deuxième critère est également intéressant puisqu'on peut constater, sur Fig. 3.18, à l'itération 75 que la priorisation du critère 2 amène à avoir des faux négatifs (courbe $\alpha_1 = 0.1$). En effet, l'impact de l'attaque sur le critère 2 n'est pas significatif, comparé au critère



(a) Évolution du critère p_1 (performance des plans).



(b) Évolution du critère p_2 (vraisemblance des routes).

FIGURE 3.17 – Résultat de la détection avec trois différentes valeurs de (α_1, α_2)

1, et donc sa priorisation entraîne que l'impact de l'attaque n'est pas non plus significatif sur le score de vraisemblance.

En conséquence, la conservation des deux critères est nécessaire puisque complémentaire comme on a pu le voir. On peut constater que ce qui amène des erreurs correspond à une trop grande priorisation d'un des critères sur l'autre. Ainsi, la pondération des critères est importante puisqu'on a vu qu'elle avait un impact sur la détection.

3.4 Conclusion

Ce chapitre complète la spécification de l'activité du contrôle présentée en développant une méthode de détection d'anomalies dans les décisions prises par le contrôleur. Tout d'abord, des propriétés structurelles définissant des conditions nécessaires à la cohérence des décisions prises par le contrôleur sont introduites. Puis, on a pu voir que ce n'était pas suffisant et donc on a proposé d'établir un modèle de l'activité de la commande. À partir de ce modèle, une méthode d'analyse de la vraisemblance des décisions est introduite. Cette méthode dépend du

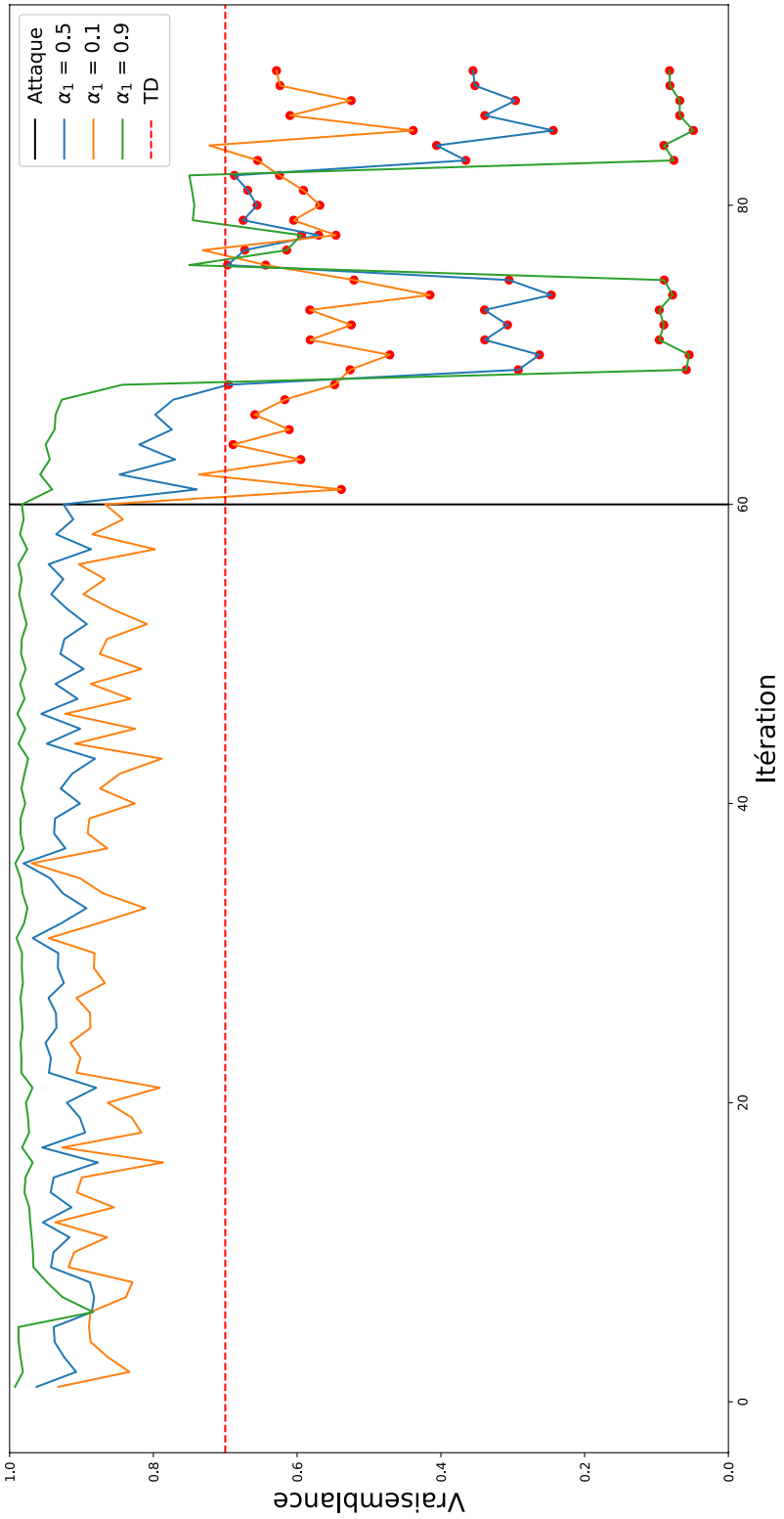


FIGURE 3.18 – Évolution du score de vraisemblance $\mathcal{L}$

type de contrôle en place : déterministe ou non. Dans le cas déterministe, nous proposons de vérifier que le contrôleur respecte les estimations des variables internes, table de routage, établies précédemment. Dans le cas d'un contrôle non déterministe, nous proposons d'établir un score de vraisemblance aux décisions prises par le contrôleur selon une approche multicritères. Les critères proposés sont liés à l'application choisie et peuvent être modifiés pour d'autres types d'applications. La méthode de détection a pour vocation de déterminer si le contrôleur est attaqué ou non. Deux critères sont introduits et on a pu voir que chacun était utile et apporte ses propres spécificités.

Dans le chapitre suivant, ces méthodes vont être mises en place sur des cas d'études afin d'évaluer les performances de nos propositions. Ces évaluations feront l'objet de discussion de nos propositions, notamment des paramètres sous-jacents à chaque modèle.

Chapitre 4

Analyse par expérimentations des solutions proposées

Dans le chapitre précédent, nous avons présenté la construction de l'observateur selon que l'algorithme de contrôle soit déterministe ou non. L'objet de cette partie consiste à évaluer les solutions ainsi obtenues afin de pouvoir discuter de leurs efficacités. Dans un premier temps, les métriques qui serviront à l'étude sont présentées en section. 4.1. Puis, nous allons voir comment réagit l'observateur lorsque le contrôleur défaille, subit une attaque de type déni de service ou bien lorsqu'un attaquant compromet le contrôleur. Dans la Table. 4.1 nous avons répertorié les différents paramètres de l'étude ainsi que le sujet des discussions selon les scénarios, les paramètres et les métriques considérées dans ce chapitre.

Paramètre d'étude	Scénario	Métriques	Section
Borne du *template* (influence de β)	Défaillance	Réactivité	Section. 4.2
	DDoS	Réactivité	Section. 4.3
	Contrôle Déterministe *Control Message Drop Attack* *Flow Rule Modification Attack*	Justesse *Recall* Précision	 Section. 4.4.1.2 Section. 4.4.1.3
Seuil TD	Contrôle Non Déterministe *Flow Rule Modification Attack*	Justesse *Recall* Précision	Critère 1 : Section. 4.4.2.1 Critère 2 : Section. 4.4.2.2
Profondeur *depth*	Contrôle Non Déterministe *Flow Rule Modification Attack*		
Les solutions d'apprentissage	Contrôle Non Déterministe *Flow Rule Modification Attack*		Section. 4.4.2.2

TABLE 4.1 – Résumé des scénarios qui vont être mis en place ainsi que l'ensemble des sujets de discussion.

Pour chacun des scénarios, la topologie de simulation sera toujours la même et est donnée en Fig. 4.1. Il y a naturellement l'infrastructure réseau, simulée avec Mininet [9] (De Oliveira et al., 2014) et le contrôleur sur une autre machine. On utilisera deux contrôleurs différents : ONOS [10] et Ryu [11] avec des algorithmes de contrôle différents. L'observateur est sur la même machine que

9. `http://mininet.org`
10. `https://opennetworking.org/onos/`
11. `https://ryu-sdn.org/`

Mininet et observe le trafic en utilisant la libraire python Scapy [12]. Il est important de rappeler que l'observateur n'envoie pas de paquets sur le réseau. Enfin, l'attaquant est sur une troisième machine (de type Kali Linux [13]).

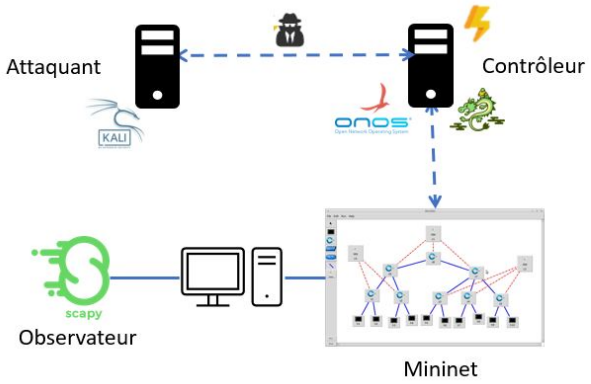


FIGURE 4.1 – Topologie physique des prochaines expériences.

4.1 Métriques d'évaluation des performances de l'observateur

Avant d'expliciter l'efficience de nos propositions de détection, nous abordons ici les différentes métriques à considérer pour notre problème de détection d'anomalies. Il s'agit de vérifier que l'observateur prend de bonnes décisions. Pour cela, on l'évaluera selon trois métriques :

1. Justesse : la proportion de bonnes décisions prises. La justesse est la proximité ou l'éloignement d'un ensemble donné de décisions par rapport à leur valeur réelle, (Šimundić, 2009). Ici, on évalue l'observateur dans sa prise de décision à tout instant (pas uniquement suite à l'attaque).
2. Précision : cela correspond au nombre de bonnes alarmes (c'est-à-dire qu'il y a bien une attaque ou une défaillance du contrôleur) rapportées au nombre total d'alarmes. (Davis & Goadrich, 2006). La précision correspond au nombre d'éléments pertinents récupérés en proportion du nombre d'éléments récupérés, c'est-à-dire $N_{Pertinents} \cup N_{Retrouve} / N_{Retrouve}$. (Buckland & Gey, 1994). On se place donc en phase d'attaque.
3. *Recall* : cela correspond au nombre de bonnes alarmes (c'est-à-dire qu'il y a bien une attaque ou une défaillance du contrôleur) rapportées au nombre de bonnes décisions qui auraient dû être prises. (Davis & Goadrich, 2006). Le *recall* est le nombre d'éléments pertinents retrouvés par rapport à l'ensemble des éléments pertinents, c'est-à-dire $N_{Retrouve} \cup N_{Pertinents} / N_{Pertinents}$. (Buckland & Gey, 1994). On se place également en phase d'attaque.

Pour évaluer ces métriques, il est nécessaire de déterminer le nombre de vrais positifs TP, vrais négatifs TN, faux positifs FP et faux négatifs FN. À partir de ces valeurs, on peut calculer :

— Justesse : $\frac{TP+TN}{TP+TN+FP+FN}$
— Précision : $\frac{TP}{TP+FP}$
— *Recall* : $\frac{TP}{TP+FN}$

12. `https://scapy.net/`
13. `https://www.kali.org/`

Ces trois métriques permettent d'évaluer la faculté de l'observateur à prendre de bonnes décisions. Il y en a deux qui se concentre sur la phase d'attaque à savoir la précision et le *recall*. La première visant à évaluer la proportion de bonnes alarmes par rapport à l'ensemble des alarmes alors que le second évalue la proportion d'alarmes levées par rapport à l'ensemble des alarmes qui aurait dû être prise. Il faut également évaluer l'observateur dans la phase nominale. C'est la raison pour laquelle on introduit également la justesse qui mesure la proportion de bonnes décisions prises globalement. Elles sont largement répandues dans le domaine de la détection d'anomalies afin d'évaluer les méthodes proposées comme on peut le retrouver dans les *surveys* suivants (Muruti et al., 2018), (Ahmed, Mahmood, & Hu, 2016) ou (Lindemann, Maschler, Sahlab, & Weyrich, 2021).

De manière générale, ces métriques sont utilisées afin d'évaluer la capacité de l'observateur à prendre de bonnes décisions. Il est également important d'étudier la réactivité de l'observateur. C'est-à-dire le temps qu'il va mettre à détecter une attaque. En raison de la structure de la méthode de détection choisie, un majorant de la réactivité correspond au maximum de l'intervalle de temps laissé au contrôleur (la borne du *template* définie dans la section. 2.4.4). Il est clair que plus l'observateur est réactif alors moins de temps l'attaquant aura pour poursuivre son attaque et endommager le réseau. Néanmoins, cette borne correspond également à une tolérance du retard du contrôleur et donc intuitivement : moins on est tolérant, plus on a de fausses alarmes et plus on est réactif. Cependant, afin de limiter le nombre de fausses alarmes, il faut être plus tolérant sur les potentiels retards du contrôleur, bien que cela aura pour conséquence d'être moins réactif. On étudiera ce compromis par la suite. Ce critère est plus répandu dans le domaine de la détection de défaillance comme dans (Fonseca et al., 2012).

Il est également important de mentionner que la réactivité ici correspond uniquement à la réactivité de la détection. Dans les perspectives des cette thèse, il faudra trouver une solution pour reprendre la main suite à une défaillance ou une attaque. Ainsi, la réactivité totale sera donc la somme de la réactivité de détection et du temps nécessaire sur la reprise de la main. Lorsque la méthode sera complète, il sera intéressant de pousser cette métrique avec l'évaluation du Mean Time To Repair (MTTR). Le MTTR sert à vérifier que la méthode ne prend pas trop de temps pour réparer un système ou un équipement ce qui n'est pas souhaitable ; car cela peut avoir un impact très négatif sur les résultats commerciaux. C'est notamment le cas pour les processus particulièrement sensibles à la défaillance et entraîne souvent des arrêts de production, des délais manqués ....

4.2 Défaillance du contrôleur

Dans un premier temps nous allons tester notre observateur en cas de défaillance du contrôleur. Le plan d'expérience est d'abord introduit et ensuite les résultats de la détection de l'observateur.

4.2.1 Scénario

La topologie de simulation considérée est celle donnée en Fig. 4.1. Le contrôleur ONOS (ONF, 2017) est chargé en utilisant une application de routage de type Dijsktra. Le contrôleur communique avec les commutateurs en utilisant OpenFlow v1.3 (ONF, Juin, 2012). De plus, le contrôleur découvre la topologie en utilisant deux applications, le fournisseur d'hôtes et Link Layer Discovery Protocol (LLDP) pour déterminer la topologie du réseau. Enfin, le module proxyARP est utilisé par le contrôleur afin de gérer les requêtes ARP sur le réseau. En ce qui

concerne la politique de routage, nous avons utilisé l'application déterministe fwd qui utilise l'algorithme de Dijkstra.

L'observateur a été implémenté sur la même machine que Mininet et la communication entre le contrôleur et les commutateurs a été capturée à l'aide de l'outil Scapy. Il est important de préciser que l'observateur n'envoie de paquets au réseau.

La topologie considérée est présentée dans la Fig. 4.2.

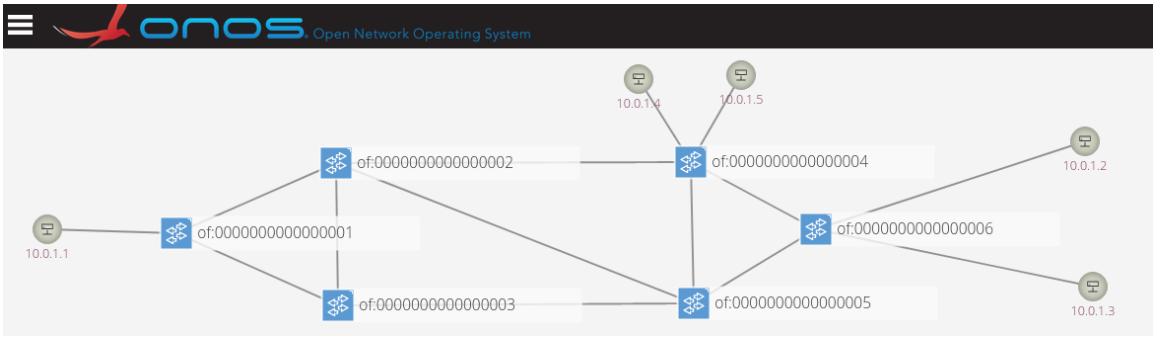


FIGURE 4.2 – Topologie réseau considérée.

En ce qui concerne les demandes du plan de données, nous avons randomisé l'interarrivée des demandes : une valeur est déterminée entre 3 et 15 s par une loi uniforme. De plus, la longueur du trafic est déterminée par le nombre d'octets à transmettre, et est également aléatoire. Nous avons explicité trois possibilités : trafic court (126 ko à transmettre, $0,1$ s est nécessaire pour la transmission), trafic moyen ($1,26$ Mo à transmettre, 10 s est nécessaire pour la transmission), et trafic plus long ($12,6$ Mo à transmettre, 100 s est nécessaire pour la transmission). L'objectif était de déterminer l'impact de la défaillance sur différents types de trafic.

La défaillance est simulée par un arrêt (*kill processus*) du contrôleur ONOS.

Maintenant, nous allons étudier les performances de l'observateur face à une défaillance du contrôleur.

4.2.2 Analyse des performances de l'observateur

Le moment de la défaillance du contrôleur est aléatoire. Les résultats de la détection sur une expérience sont présentés sur la Fig. 4.3. Sur la figure, les décisions de l'observateur sont représentées par les points suivants : zéro s'il n'y a pas d'alarme et un sinon. De la même façon, l'état de la défaillance est représenté par la courbe dont la valeur vaut : zéro s'il n'y a pas de défaillance et un sinon.

Il est explicite que la défaillance est détectée par l'observateur. En effet, la défaillance a pour effet de stopper l'envoi des messages depuis le contrôleur. En conséquence, son délai de réponse tend vers $+\infty$ et il ne peut pas envoyer de commande avant la limite fixée lors de l'établissement du *template*.

Rappelons que nos métriques sont de deux types : évaluation de l'exactitude des décisions du contrôleur et ensuite de la réactivité de l'observateur. Cependant, en cas de défaillance du contrôleur, comme celle présentée en Fig. 4.3, l'évaluation de l'exactitude des décisions de l'observateur est évidente puisqu'on analyse l'activité du contrôle et non l'état du contrôleur. En ce sens en cas de défaillance, le seul moyen que l'observateur ne lève pas une alarme est que la défaillance soit corrigée juste avant la fin du temps laissé par l'observateur dans le *template*. Auquel cas, le contrôle sera rendu dans les limites du tolérable, ce qui n'est pas un souci vis-à-vis du réseau. En

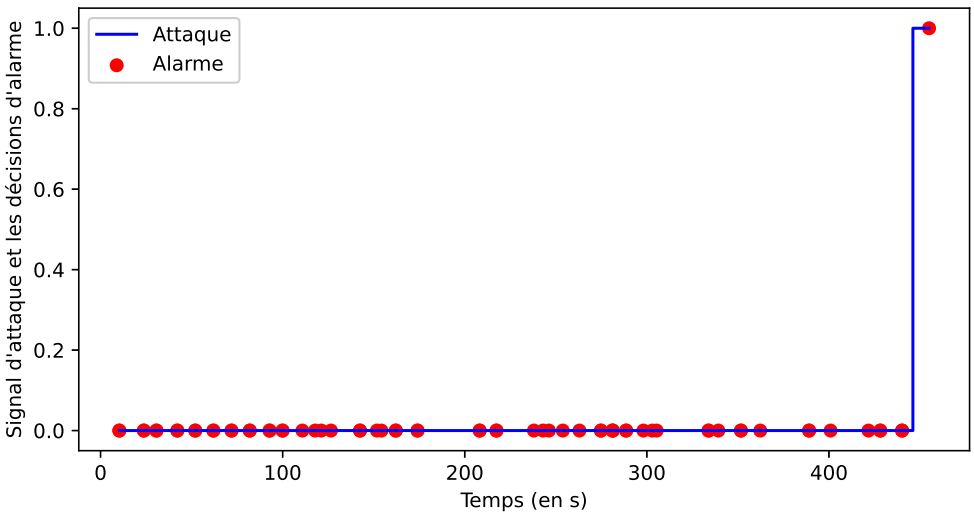


FIGURE 4.3 – Évolution des alarmes dans le cas d'une défaillance.

effet, il peut y avoir des défaillances de différents niveaux d'intensité qui touche le contrôleur. Par exemple une simple coupure de courant n'interrompt pas définitivement le contrôleur. Ce type de défaillance n'est pas discuté ici et seule une défaillance franche du contrôleur sera considérée. La suite vise donc à étudier l'impact de la tolérance laissée à la borne que l'on se fixe.

L'expérience décrite précédemment a été répétée 35 fois. La défaillance est bien détectée à chaque fois. Ce sera le cas pour toute défaillance (car le retard tend vers $+\infty$). Ainsi, la précision et le *recall* de l'expérience présentée dans la Fig. 4.4, dépendent uniquement de la phase avec attaque. Ici, comme le délai en cas d'échec tend vers $+\infty$, tous les défaillances sont détectées, ce qui signifie que le *recall* est égal à 1. Néanmoins, la précision augmente avec β en raison de l'augmentation de la tolérance. En effet, on constate que la précision augmente avec β en raison du nombre de faux positifs. Cela signifie que la tolérance est extrêmement stricte, avec $\beta = 1,2$ contre $\beta = 2$. Ainsi, par rapport à $\beta = 2$, $\beta = 1,2$ est sensible à l'évolution du trafic, et donc, à l'évolution du temps de réponse du contrôleur. Ainsi, en général, si β augmente, alors le taux de faux positifs diminue du fait de l'augmentation de la tolérance. Cet effet se ressent donc sur la justesse.

Effectivement, observons l'évolution de la réactivité en fonction de β. La courbe est donnée en Fig. 4.5.

Comme prévu, la réactivité augmente avec β. Tout d'abord, la réactivité avec un facteur de tolérance de $\beta = 1,2$ est concentré sur 5 ms en raison du manque de tolérance à toute évolution du trafic. Ensuite, la réactivité s'étale avec la tolérance (et donc β).

Plus précisément, on observe un impact de Poisson sur la courbe, en fonction du paramètre β. Tout d'abord, concentrons-nous sur l'évolution du maximum des courbes. Globalement, l'évolution des maximums des courbes données en Fig. 4.5 diminue exponentiellement avec β (suivant une loi de l'ordre de $e^{-n\times\beta}$ où n est un coefficient définissant le profil de la courbe). En Fig. 4.6 nous avons tracé l'évolution théorique des maximums de la fonction de masse d'une loi de Poisson en fonction du paramètre λ comparé aux maximums des fonctions de masse de la réactivité de l'observateur de paramètre β.

On suppose que le paramètre β est lié au paramètre de la loi de Poisson que suivrait la

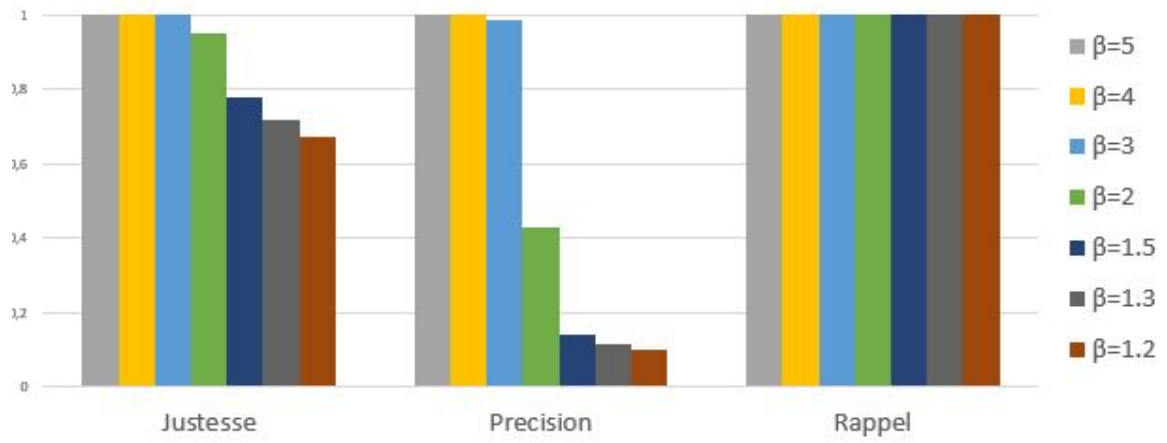


FIGURE 4.4 – La valeur des métriques pour les différentes valeurs de β en cas de défaillance du contrôleur.

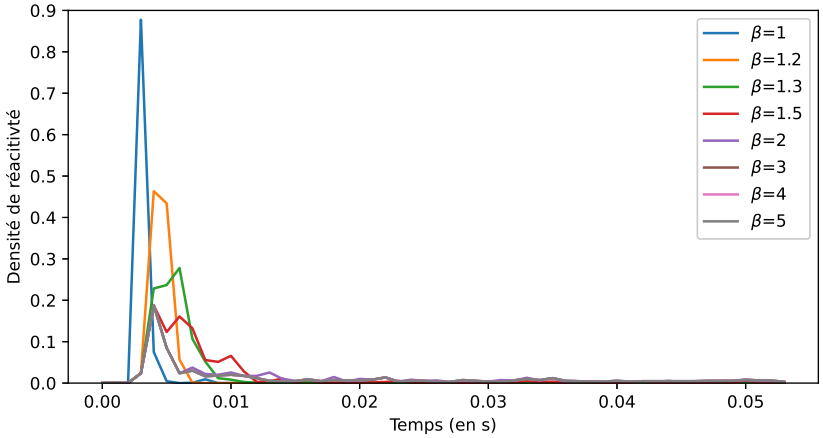


FIGURE 4.5 – Réactivité de l'observateur en fonction de β dans le cas de défaillance du contrôleur.

réactivité de l'observateur sans que l'on ait à ce stade formalisé le lien et on supposera donc que c'est une loi de paramètre β. Pour l'amplitude, on a déterminé le maximum pour plusieurs valeurs de β : 0,1 ; 1,2 ; 1,3 ; 1,5 ; 2 ; 2,5 et 5. Et on a tracé l'évolution de ce maximum pour une loi de Poisson classique. On peut voir que l'amplitude observée évolue selon la même tendance que l'amplitude d'une loi de Poisson théorique, ce qui consolide nos hypothèses.

Ensuite, étudions l'élargissement de la courbe c'est-à-dire la durée de l'intervalle de temps pour lequel 95% des valeurs de réactivité sont comprises. Ici, on a déterminé cette durée pour les valeurs de β suivantes : 1,2 ; 1,3 ; 1,5 ; 2 ; 2,5 ; 3 ; 3,5 ; 4 ; 4,5 et 5 et on tracé son évolution en Fig. 4.6. On se rend compte que cette durée augmente linéairement, comme représenté en Fig. 4.7, ce qui est également le cas pour une distribution de Poisson. Nous n'avons pas identifié et démontré la source de cet effet de Poisson, mais ceci est probablement dû au fait que la demande des infrastructures est suffisamment espacée pour que le temps de réponse du contrôleur soit indépendant du temps écoulé depuis le dernier événement. Ainsi, le temps de réponse du contrôleur semble suivre une distribution de Poisson et en conséquence la réactivité de l'observateur aussi.

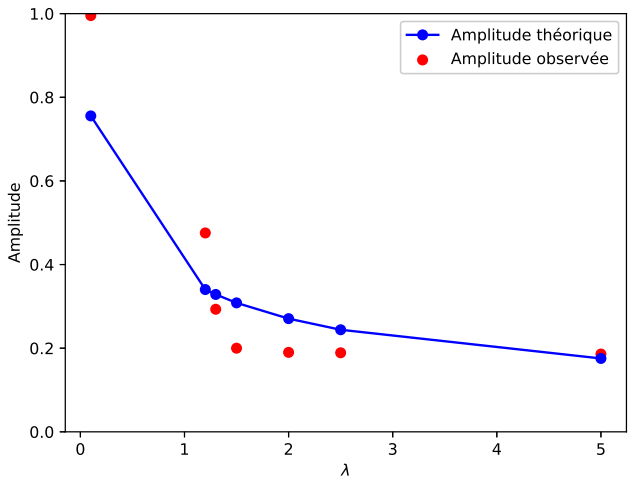


FIGURE 4.6 – Évolution du maximum de l'amplitude des lois de Poissons par rapport à nos points.

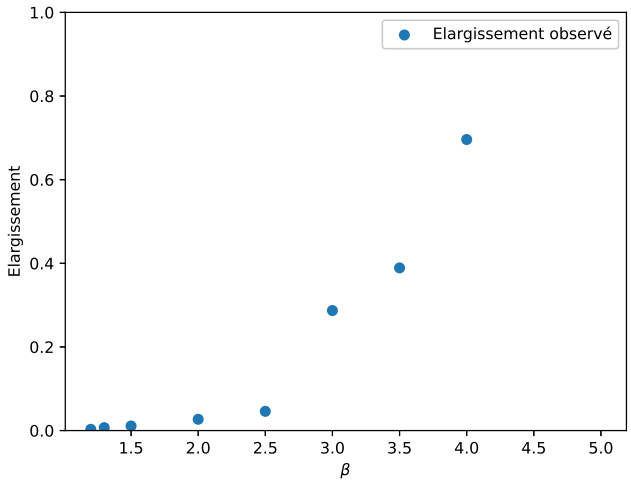


FIGURE 4.7 – Évolution de l'étendue des lois de Poisson par rapport à nos points.

4.3 Étouffement du contrôleur

Dans cette section nous allons voir la réaction de l'observateur lors d'attaque de type déni de service distribué visant le contrôleur. La conséquence de cette attaque est l'étouffement de ce dernier (qui dans le cas extrême s'apparentera à une défaillance du contrôleur).

4.3.1 Scénario type déni de service distribué

Pour rappel, la topologie de simulation est représentée sur Fig. 4.1.

L'objectif de l'attaque est de lancer une attaque de type déni de service distribué (DDoS) sur le contrôleur. L'outil utilisé pour lancer cette attaque est le logiciel `hping3`. L'idée est d'inonder

le contrôler de requêtes TCP de type SYN (qui sont des demandes d'établissement de connexion).

Les effets de l'attaque sont représentés sur la Fig. 4.8. Il y a la courbe représentant le statut de l'attaque : 0 si pas d'attaque et 1 sinon. Ensuite, les points représentent les temps de réponse du contrôleur observé par l'observateur. On peut voir que l'attaque a effectivement pour effet de ralentir le contrôleur avec une augmentation significative de ce temps de réponse jusqu'à 20 secondes (soit près de 20 fois plus que dans le cas sans attaque).

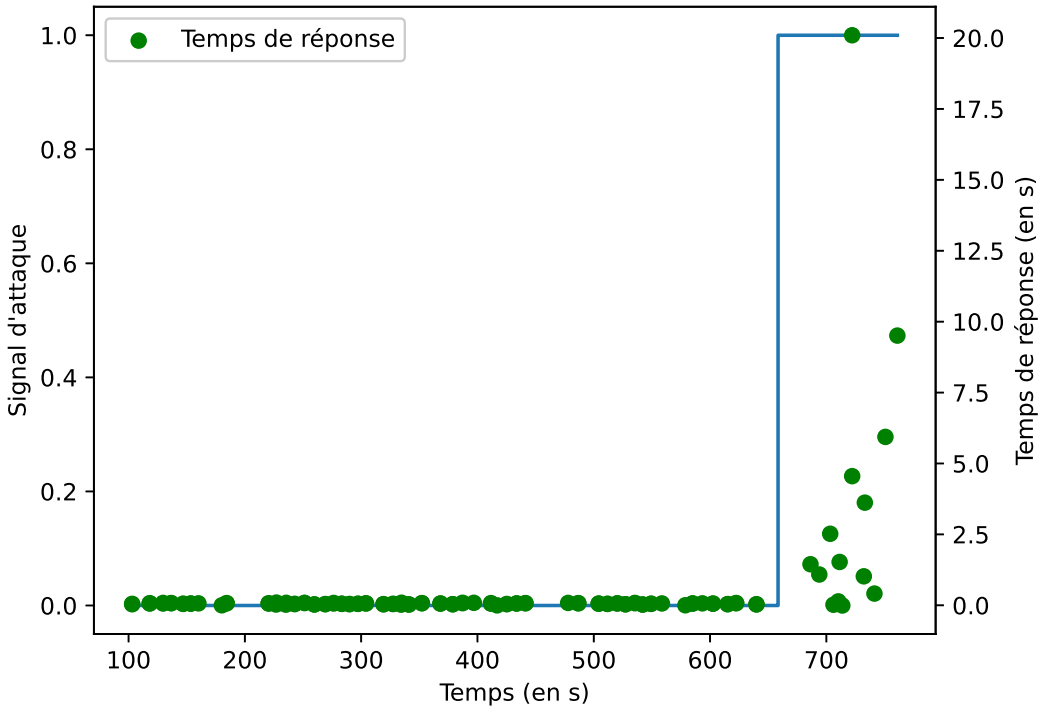


FIGURE 4.8 – Évolution du temps de réponse du contrôleur

Nous allons mettre en situation l'observateur pour voir son comportement lorsque le contrôleur subît des attaques de type DDoS.

4.3.2 Analyse des performances de l'observateur

4.3.2.1 Cas d'une DDoS simple

Dans un premier temps, nous allons voir comment l'observateur réagit lorsque le contrôleur subit une DDoS classique. Nous avons représenté les résultats de l'observateur selon le facteur de conservatisme β choisi. Pour $\beta = 2$ les résultats sont représentés sur Fig. 4.9a, pour $\beta = 1.3$ les résultats sont sur Fig. 4.9b et pour $\beta = 1$ les résultats sont sur Fig. 4.9c. Sur les trois figures, on retrouve une courbe en orange représentant le moment de l'attaque et un nuage de points représentant les alarmes de l'observateur (alarme lorsque le point est à 1 et l'absence d'alarme est représentée par un point à 0). On peut voir que plus le facteur de conservatisme β augmente plus le nombre de fausses alarmes (une fausse alarme signifie que l'observateur a levé une alarme alors qu'il n'y a pas d'attaque) diminue comme cela peut se voir en comparant les graphes pour

$\beta = 2$ et $\beta = 1$ sur les Fig. 4.9a et 4.9c respectivement. Il se peut dans la phase nominale que le contrôleur ait du retard (à cause d'un autre processus utilisant plus de ressources du processeur ou à cause d'une requête plus longue que les autres par exemple) et donc β permet ainsi de se prémunir face à l'incertitude de l'apparition et l'intensité de ce genre de retard.

Par ailleurs, dans ce cas la DDoS est brutale donc on va détecter l'attaque, quel que soit le paramètre β ici. Or, en réalité, selon l'intensité de cette attaque, l'observateur peut être mis en échec et c'est ce que nous allons voir ci-après.

4.3.2.2 Cas de DDoS d'intensités multiples

Les résultats précédents sont faits en cas d'une DDoS intense et profondément significative. Il existe diverses façons de mettre en place des attaques de type déni de service comme présenté dans (Pascoal, Fonseca, & Nigam, 2020) ou (Vaccari, Aiello, & Cambiaso, 2020). La variation de l'intensité de l'attaque est utilisée par les attaquants afin de ne pas se faire détecter. Nous allons donc tester l'observateur face à différentes intensités d'attaque de type DDoS.

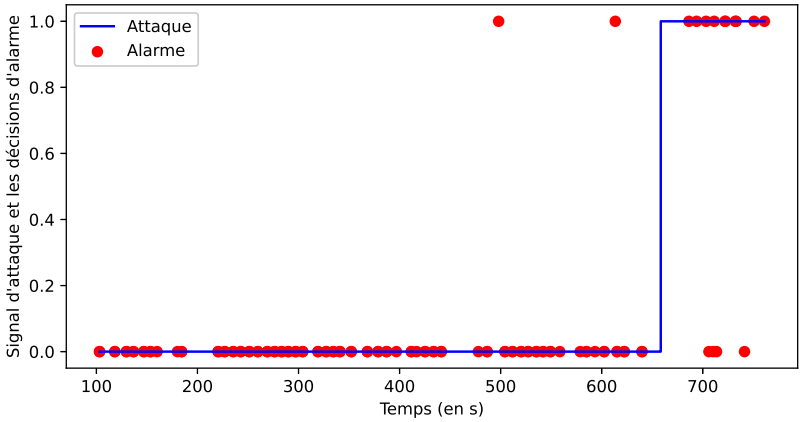
Cette fois on va lancer trois attaques DDoS, chacune d'intensité différente. Pour cela, la durée de l'attaque est modulée : la première attaque dure 10 secondes, la suivante 100 secondes et enfin la dernière 1000 secondes.

Le temps de réponse du contrôleur est représenté en Fig. 4.10. On peut voir en premier lieu l'attaque courte qui a un léger impact sur le contrôleur, proportionnel à l'intensité de l'attaque. Puis, la deuxième attaque provoque une augmentation significative du temps de réponse du contrôleur jusqu'à engendrer un temps de réponse de 10s. Cependant, bien que l'impact de l'attaque soit significatif, elle ne fait pas défaillir le contrôleur. Contrairement à la dernière attaque. Le graphe s'arrête au moment où le contrôleur défaille.

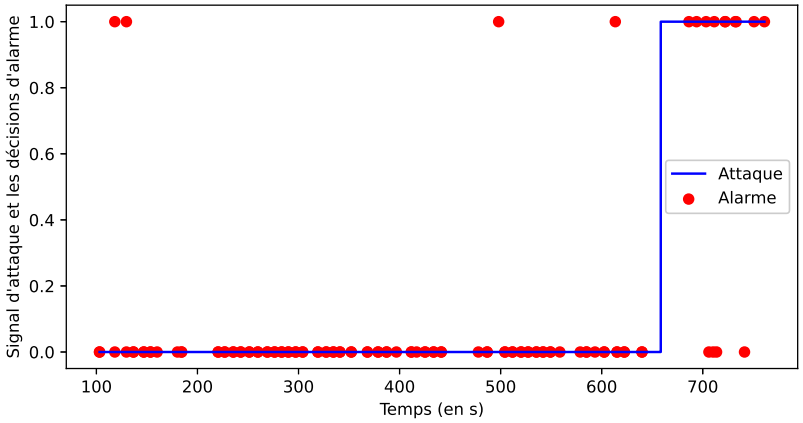
On peut voir la réponse de l'observation pour $\beta = 1,2$ en Fig. 4.11a, pour $\beta = 2$ en Fig. 4.11b et pour $\beta = 5$ en Fig. 4.11c.

En première approche, on peut voir que pour $\beta = 1,2$ il y a énormément de fausses alarmes. En effet, cela est dû au manque de flexibilité face à l'incertitude du retard du contrôleur comme décrit dans la section plus haut. Cependant, on peut se rendre compte également qu'une trop grande flexibilité peut amener à des situations indésirables et la non-détection des DDoS de faible intensité malgré un retard non négligeable du contrôleur comme observé sur Fig. 4.10. Par exemple pour $\beta = 5$ on peut voir que l'observateur est tellement tolérant que les DDoS plus courtes ne sont pas détectées. Cette non-détection peut avoir des conséquences indésirables pour les prochaines détections. En effet, un retard dû à une attaque, mais qui n'est pas détecté a pour effet d'augmenter l'incertitude sur le temps de réponse du contrôleur. De par l'approche proposée, cette incertitude aura pour incidence d'augmenter la limite de temps laisser par l'observateur (c'est-à-dire d'augmenter la valeur de la borne). De plus, cette augmentation implique l'acceptation de plus grand délai et par un processus de récurrence, cette borne peut tendre vers $+\infty$ et donc empêcher la détection d'une attaque qui a un impact de plus en plus significatif.

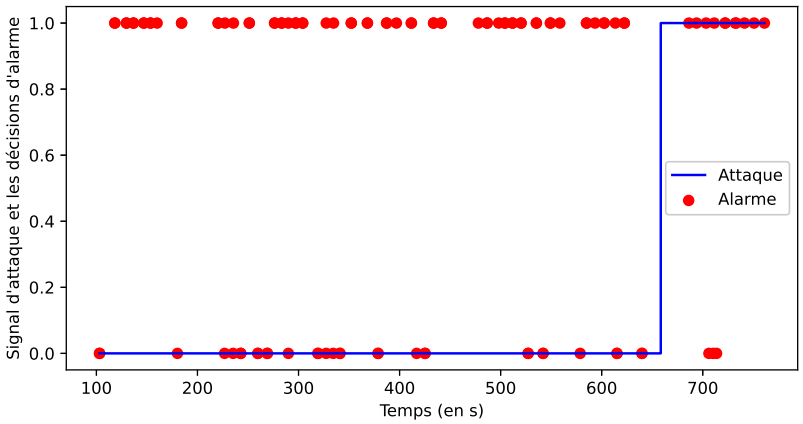
Nous allons développer ce phénomène mathématiquement. Considérons un contrôleur non fautif. On rappelle que la borne est calculée à partir d'un ensemble de mesures des temps de réponse du contrôleur jugées valide Lt_{Rep}. Parmi cet ensemble la borne est fixée comme une proportion du pire cas observé $t_{Nominal} = max(Lt_{Rep})$ tel que $t_{borne} = \beta \times t_{Nominal}$ où β est le facteur de protection. On supposera $\beta > 1$ pour la suite. Ceci étant dit, considérons le lancement d'une attaque de type déni de service. Le premier temps de réponse retardé du contrôleur est $t_{Attaque,1} = \beta \times t_{Nominal}$. Il est dans la limite et est donc jugé comme valide par l'observateur ce



(a) Nuage de points pour le cas où le facteur de protection est 2.



(b) Nuage de points pour le cas où le facteur de protection est 1.3.



(c) Nuage de points pour le cas où le facteur de protection est 1.

FIGURE 4.9 – Réaction de l'observateur face à un DDoS visant le contrôleur pour trois facteurs β différents.

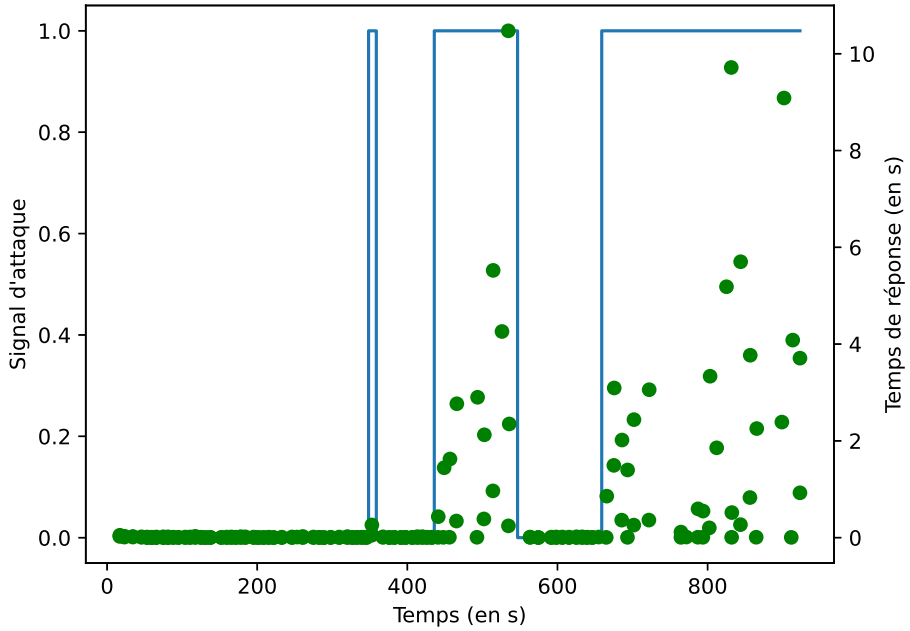


FIGURE 4.10 – Temps de réaction du contrôleur suite aux différentes attaques DDOS.

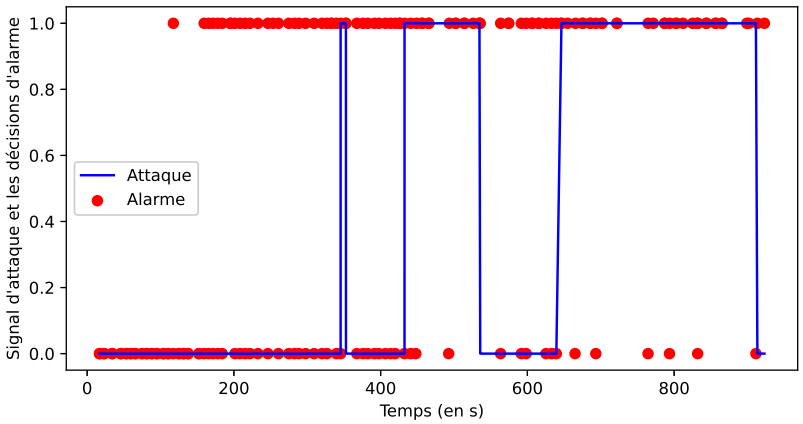
qui implique la fixation d'une nouvelle borne $t_{borne} = \beta \times t_{Attaque,1} = \beta^2 \times t_{Nominal}$. Le second retard du contrôleur est une nouvelle fois à la limite $t_{Attaque,2} = \beta^2 \times t_{Nominal}$ ce qui amène la fixation à $t_{borne} = \beta \times t_{Attaque,2} = \beta^3 \times t_{Nominal}$. Par récurrence, à la suite du n-ième retard la borne sera fixée à $t_{borne} = \beta^n \times t_{Nominal}$ et comme $\beta > 1$, la suite géométrique β^n tend vers $+\infty$. Le cas qui vient d'être formalisé correspond au pire cas et à titre illustratif en Fig. 4.12 l'évolution du facteur de protection en fonction de n est représentée pour différentes valeurs de β. Très clairement, on se rend compte que le pire cas décrit est vrai pour tout β, mais que plus ce facteur est grand, plus l'incertitude augmente rapidement.

C'est la raison pour laquelle, nous avons proposé l'introduction d'une limite de temps t_{lim} correspondant à une contrainte applicative du temps de réponse du contrôleur. Et le *timeout* sera donc le minimum entre la borne tolérée (calculé en fonction du pire cas) et cette limite applicative. Par ailleurs, cette limite provient de la méthode de calcul qui se concentre uniquement sur le pire cas, sans prendre en considération les autres valeurs du jeu de données. Ainsi, en perspective de cette proposition, on pourrait fixer la borne selon d'autres méthodes et notamment en nous inspirant des méthodes similaires au calcul du temps de transmission maximal (RTO) du protocole TCP (Jacobson, 1988).

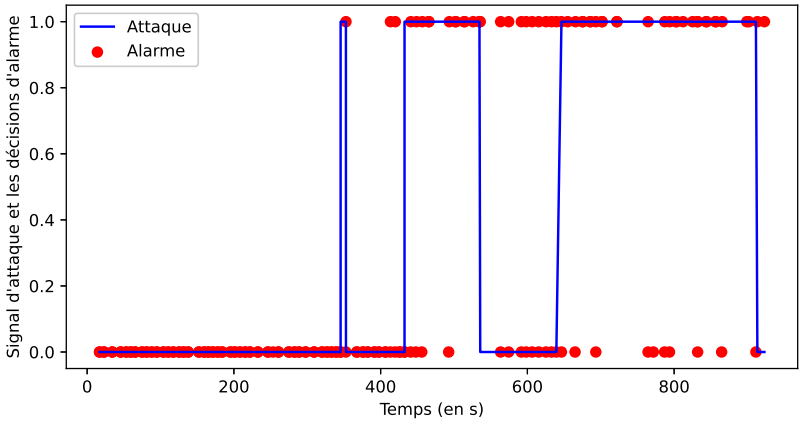
4.3.2.3 Discussion sur l'effet de la borne temporelle

L'expérience qui va être mise en place a pour but de discuter sur des effets de la borne temporelle de l'observateur face à des DDoS d'intensités différentes. Pour cela, on n'injectera régulièrement des DDoS, en modulant la longueur de la DDoS et on observera les réactions de l'observateur au comportement du contrôleur.

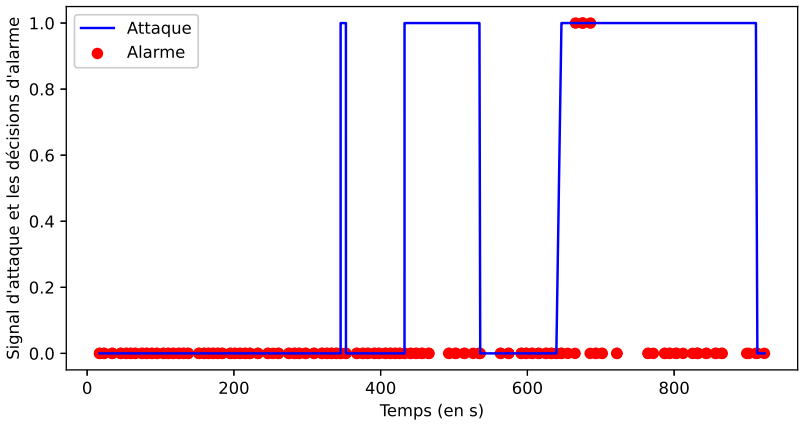
La durée de l'attaque est désormais choisie aléatoirement suivant une loi uniforme entre 20 et 200 secondes. Le but est d'affecter le comportement du contrôleur, mais pas de le rendre



(a) Réaction de l'observateur avec $\beta = 1,2$ face aux trois DDoS décrites.



(b) Réaction de l'observateur avec $\beta = 2$ face aux trois DDoS décrites.



(c) Réaction de l'observateur avec $\beta = 5$ face aux trois DDoS décrites.

FIGURE 4.11 – Réaction de l'observateur face à trois DDoS, visant le contrôleur, d'intensités différentes pour trois facteurs β différents.

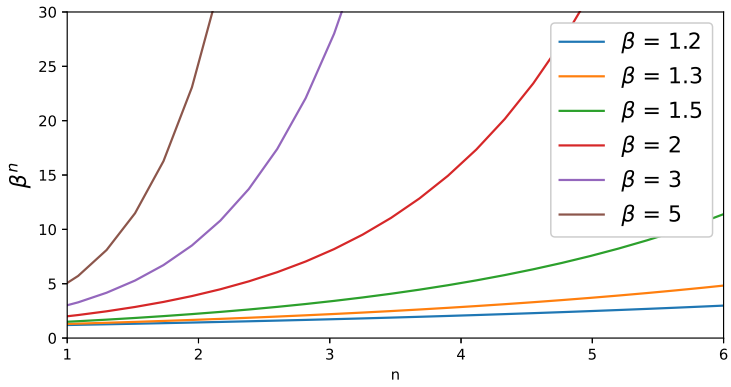


FIGURE 4.12 – Évolution du facteur de protection β dans le pire cas.

totalement défaillant pour continuer l'expérience. Rappelons que nous avons vu en Fig. 4.10 que le contrôleur peut ne pas être énormément retardé pour des attaques courtes alors qu'il l'est pour les attaques de durée moyenne.

En parallèle, le trafic Mininet est également aléatoire. Il y a des flux aléatoires avec une inter-arrivée des paquets aléatoire. Tout d'abord la longueur du flux, on joue sur le nombre de données à transmettre : trafic court (126 ko à transmettre, 0,1 s est nécessaire pour la transmission), trafic moyen (1,26 Mo à transmettre, 10 s est nécessaire pour la transmission), et trafic long (12,6 Mo à transmettre, 100 s est nécessaire pour la transmission). De plus, l'interarrivée est choisie uniformément entre 3 et 15 secondes pour accélérer le trafic aléatoirement.

Les valeurs des métriques sont données en Fig. 4.13

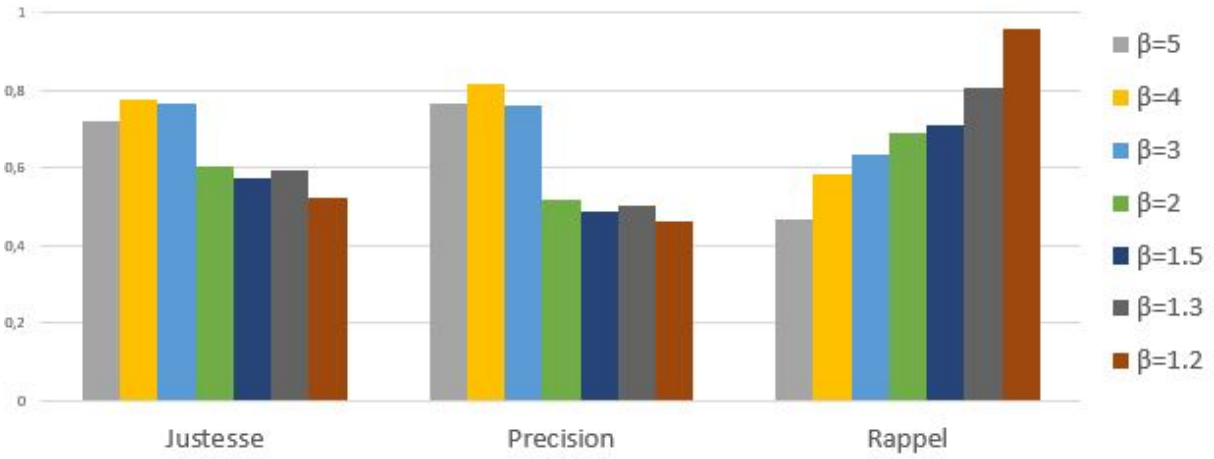


FIGURE 4.13 – Métrique pour différentes valeurs de β en cas d'attaque et trafic aléatoire.

Concentrons-nous sur la précision. On constate que la précision augmente avec β en raison du nombre de faux positifs comme on peut l'observer en comparant la Fig. 4.11a et la Fig. 4.11b. Cela signifie que la protection laissée en cas d'incertitude est trop faible avec $\beta = 1,2$ par rapport à $\beta = 2$. Ainsi, $\beta = 1,2$ est sensible, par rapport à $\beta = 2$, à l'évolution du trafic et donc à l'évolution du temps de réponse du contrôleur. Ainsi, en général, si β augmente, le taux de faux positifs diminue en raison de la limite de temps qui augmente également.

Cependant, être conservateur a un aspect négatif en matière de détection. En effet, augmenter la valeur β implique d'augmenter la tolérance au retard du contrôleur et donc les attaques ne sont pas détectées tant que l'effet n'est pas supérieur à la limite fixée. En effet, considérons l'attaque de 10 secondes : avec $\beta = 5$ l'attaque n'est pas détectée comme on peut le voir sur la Fig. 4.11c. Ce qui signifie que l'augmentation de β conduit à la diminution du taux de vrais positifs.

Ces effets ont un impact similaire sur la justesse.

Considérons maintenant le *recall*. Cette métrique correspond au nombre d'attaques détectées par rapport au nombre d'attaques qui auraient dû être détectées. Comme développé plus haut, le taux de vrais positifs diminue lorsque β augmente. Pour les mêmes raisons, le taux de faux négatif diminue linéairement lorsque β augmente. Ensuite, le *recall* est inversement proportionnel à β comme observé sur la Fig. 4.13. Par conséquent, on peut conclure que la borne temporelle a un impact et qu'une valeur plus élevée de β implique une diminution du nombre d'attaques détectées, mais réduit le nombre de faux positifs (c'est-à-dire de fausses alarmes).

La réactivité est considérée comme le temps de réaction de l'observateur à une requête d'un commutateur. Cette réaction est l'observation d'un chemin du contrôleur ou d'un timeout. Ainsi, nous considérons tous les points des expériences présentées juste au-dessus (les 1933 points) et déterminons la densité de la réactivité du contrôleur. Pour déterminer la densité, nous avons proposé de diviser l'intervalle $[0, 52]$ ms (en 52 intervalles). La limite de cet intervalle, 52 ms, a été choisie, car 99% des timeout sont inférieurs à $0,052$ pour $\beta = 1,2$, $\beta = 1,3$ et $\beta = 1,5$ et 90% pour $\beta = 2$.

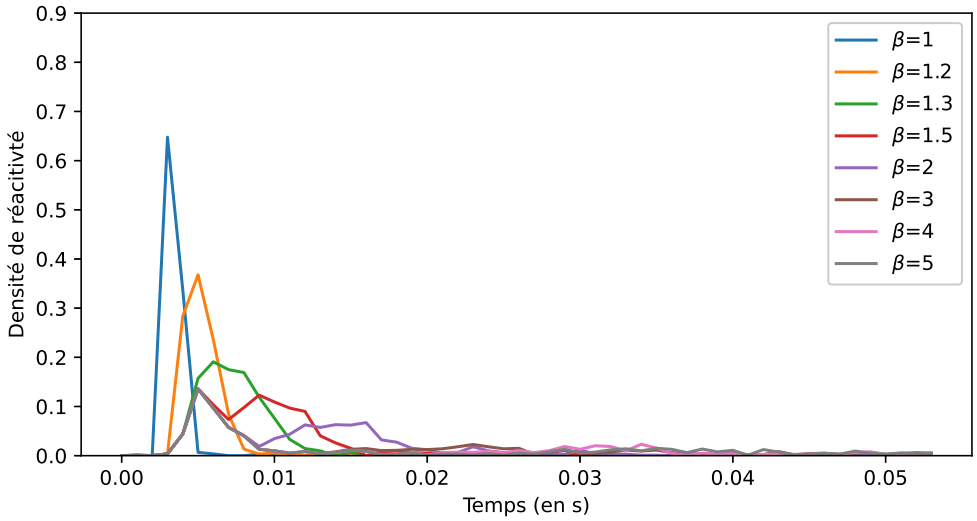


FIGURE 4.14 – La densité de réactivité de l'observateur selon β dans le cas d'attaque de déni de service d'intensité aléatoire sur le contrôleur.

Le résultat de la réactivité est représenté sur la Fig. 4.14. Comme prévu, la réactivité augmente avec β. Tout d'abord, la réactivité avec un facteur de protection de $\beta = 1,2$ est concentrée sur 5 ms en raison du manque de tolérance à toute évolution du trafic. Ensuite, la réactivité augmente avec la borne temporelle (et donc β) même s'il y a la même augmentation jusqu'à $t = 5$ ms. Cette hausse est due au fait que les paquets sont cohérents pour la borne temporelle définie avec $\beta = 1.5$ donc ils le sont pour une valeur supérieure de β. Ensuite, les courbes évoluent différemment et on observe un profil de Poisson sur la courbe similaire à celui observé lors de

l'évaluation de la réactivité dans le cas de la défaillance. On peut néanmoins constater que la réactivité est différente durant les deux expériences et on allons les comparer dans la partie qui suit.

4.3.2.4 Comparaison avec le cas de défaillance

On peut constater que les conclusions sur l'observateur sont similaires lorsqu'il est confronté à un cas de défaillance du contrôleur et à l'étouffement du contrôleur notamment par une attaque de type déni de service. Cela valide notre méthode dans le sens où on détecte des problèmes de sûreté et de sécurité de la même manière sans les distinguer puisqu'on se concentre uniquement sur les conséquences de l'attaque.

Il y a néanmoins une différence dans le comportement de la réactivité. On se rend compte que si le maximum des courbes est relativement similaire, ce n'est pas le cas pour la durée de l'intervalle de temps qui comprend 90% des valeurs. Une comparaison de la durée de ces intervalles en fonction de β est donnée dans la Table. 4.2.

	Durée de l'intervalle	
β	Défaillance	DDoS
1	0.001	0.001
1.2	0.003	0.003
1.3	0.007	0.007
1.5	0.014	0.011
2	0.135	0.168
3	0.148	0.287
4	0.149	0.699
5	0.15	1.218

TABLE 4.2 – Comparaison de la durée de l'intervalle de temps qui comprend 90% des réactivités de l'observateur dans le cas d'une défaillance ou d'un DDoS en fonction du paramètre β.

La défaillance considérée est brutale, ce qui entraîne que le contrôleur n'est pas étouffé et donc, avant la défaillance, il n'y a pas de retard particulier. Par conséquent, l'évolution de l'incertitude n'est liée qu'à l'évolution du trafic du flux en cas de défaillance. Contrairement au cas de DDoS dont l'incertitude est polluée par des retards tolérés.

En réalité, cette différence dans les résultats s'explique par l'intensité des attaques qui étaient différentes de celle de la défaillance mise en place. En effet, la défaillance considérée est brutale et ne pollue donc pas l'incertitude sur la latence du contrôleur contrairement à des retards liés à une DDoS. Ce ne serait pas le cas avec des défaillances d'intensité moins importantes (comme par exemple une coupure de courant légère). Et étant donné que l'observateur détecte des anomalies sans distinguer si elles sont causées par des problèmes liés à la sûreté ou la sécurité on peut s'attendre à ce que l'observateur se comporte exactement comme si c'était des attaques de type déni de service d'intensité plus faible dont les résultats sont donnés ci-dessus.

4.4 Compromission du contrôleur par un attaquant

Les sections précédentes ont présenté la réaction de l'observateur face à deux types de menaces de la couche contrôle : un arrêt brutal ou un étouffement du contrôleur. Les conséquences sont

respectivement de supprimer brutalement le contrôle sur le réseau et de ralentir l'acheminement des décisions de contrôle. Ici, nous allons voir le troisième type de menace présenté dans le chapitre 1 : compromission du contrôleur par un attaquant. Cette fois, la conséquence est de pouvoir mettre en place un contrôle différent de ce qui était prévu.

4.4.1 Dans le cas d'un algorithme de contrôle déterministe

Nous allons voir la réaction de l'observateur lorsque le contrôle nominal est déterministe, en appliquant l'Algo. 6. Pour rappel, on considère qu'un algorithme de contrôle est déterministe lorsque étant donné une requête particulière (sans évolution de topologie) alors l'algorithme produira toujours la même décision.

4.4.1.1 Scénario

La topologie de simulation est présentée en Fig. 4.1. L'attaquant est sur une machine Kali Linux [14] et nous allons utiliser le logiciel suivant : Metasploit [15]. Metasploit est un projet en relation avec la sécurité des systèmes informatiques. Son but est de fournir des informations sur les vulnérabilités de systèmes informatiques. Le plus connu des sous-projets est le Metasploit Framework, un outil pour le développement et l'exécution d'exploits (logiciels permettant d'exploiter à son profit une vulnérabilité) contre une machine distante. Les exploits sont des « programmes » exploitant des failles, dans le but de réaliser une action malveillante sur une cible (prise de contrôle, copie de fichiers, etc ...). La faille logicielle que nous allons utiliser ici est celle de l'application FTP : vsftpd v. 2.3.4. Elle permettra à l'attaquant de compromettre le contrôleur et de mettre en place diverses attaques (Lee et al., 2017a) par modification des fichiers de la machine du contrôleur puisque l'attaquant y a accès.

La topologie considérée est celle donnée en Fig. 4.15.

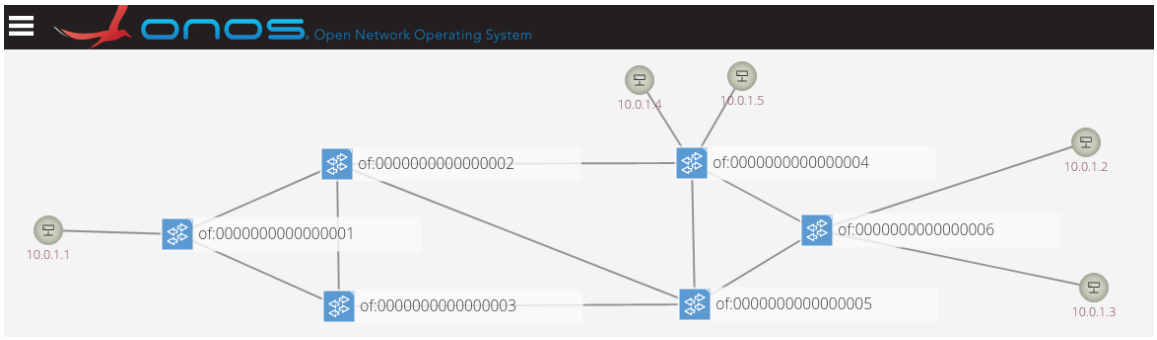


FIGURE 4.15 – La topologie mise en place pour tester l'observateur face à une attaque sur un contrôle déterministe.

4.4.1.2 *Control Message Drop Attack*

Cette attaque a été définie par (Lee et al., 2017a), *Control Message Drop Attack*, et l'objectif de cette attaque est de supprimer les paquets envoyés par le contrôleur, mais seulement ceux en direction du switch 4. Le but étant de montrer que malgré l'attaque du contrôleur, s'il rend un

14. https ://www.kali.org/
15. https ://www.metasploit.com/

service correct (chemin non concerné par le switch 4) alors l'observateur ne jugera pas qu'il y ait un problème.

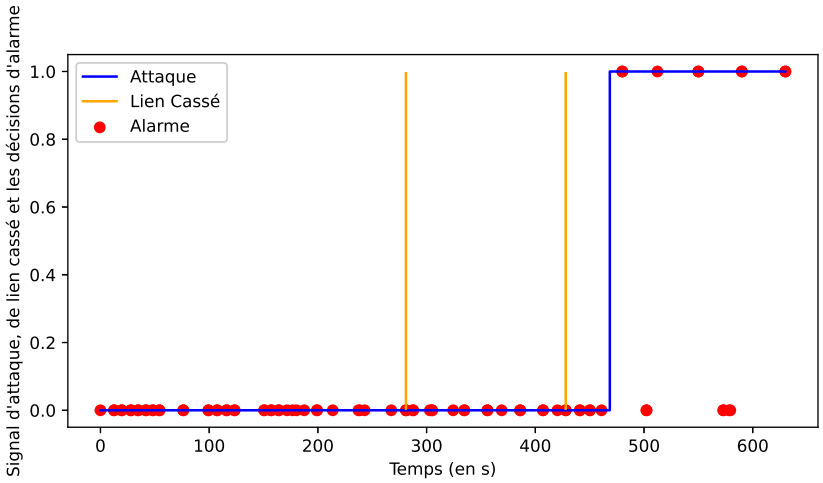


FIGURE 4.16 – Résultat pour l'attaque de suppression de paquet

Les résultats de ce cas sont représentés sur la Fig. 4.16. Il y a en orange la courbe représentant le moment de l'attaque : à 0 sans attaque et à 1 avec attaque et en bleu, les points représentant les décisions de l'observateur : lorsque le point est à 0 alors pas d'alarme et si à 1 alors alarme. Pour vérifier que le contrôleur réagissait bien lorsqu'il y avait une évolution de la topologie des infrastructures, nous avons cassé deux liens à deux moments différents et ces instants sont représentés par la courbe grise.

On peut voir que durant la phase nominale (courbe orange à 0), il n'y aucune fausse alarme. C'est dû au fait que le contrôle soit déterministe, l'observateur ne se trompe pas. On peut observer que c'est le cas aussi suite aux ports status (qui signalent la rupture des liens), la réaction du contrôleur a été bien analysée par l'observateur.

Après l'attaque (courbe orange à 1) on peut voir des faux négatifs et de vrais positifs. En ce qui concerne les vrais positifs, ils sont détectés par l'expiration du *template* pour les paquets censés être à destination du switch 4. Les faux négatifs s'expliquent par le fait que l'attaque vise seulement un switch donc les chemins non liés à ce switch sont corrects, bien installés par le contrôleur et donc l'observateur juge que l'activité du contrôle est correcte, ce qui est le cas pour ces chemins. En effet, l'observateur ne cherche pas à évaluer l'état du contrôleur, mais l'état de l'activité de la commande. On assume ici de laisser le contrôleur infecté tant qu'il rend son service et que le contrôle est sans anomalie.

4.4.1.3 *Flow Rule Modification Attack*

Nous allons maintenant passer à une seconde attaque visant à d'autres types de conséquences sur le réseau. Encore une fois présentée dans (Lee et al., 2017a), il s'agit de la *Flow Rule Modification Attack*. Cette fois, l'attaquant va détériorer le contrôle afin de réduire la qualité de service au niveau des infrastructures. Pour cela, il va mettre en place une attaque de type modification des variables internes du contrôleur. Plus précisément, les paquets en direction du switch 4 sont retransmis sur un mauvais port. De plus, comme précédemment, deux liens sont cassés durant l'expérience. Les résultats sont représentés en Fig. 4.17.

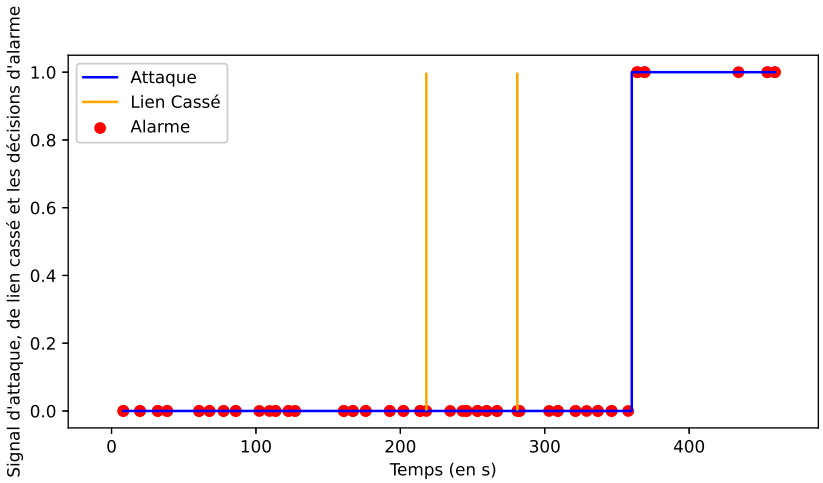


FIGURE 4.17 – Résultat pour l'attaque de modification interne des variables du contrôleur.

Comme précédemment, lors de la phase nominale il n'y a pas de fausse alarme. Cela est toujours dû au comportement déterministe du contrôleur qui n'amène pas d'évolution ou d'aléa dans le contrôle pour l'observateur. On peut également noter que l'observateur a correctement analysé la réaction du contrôleur suite à la cassure d'un lien sur le réseau. Ainsi, de manière générale lors de la phase nominale on n'observe aucune fausse alarme du contrôleur. Ensuite, lorsque le comportement fautif se manifeste suite à l'attaque, alors l'observateur le détecte de nouveau correctement.

On n'évalue pas la réactivité de l'observateur puisque cette réactivité est majorée par la borne du *template* comme étudiée dans la section précédente.

En conclusion, l'observateur réussit à détecter la compromission du contrôleur dans le cas d'un contrôle déterministe. La détection est la même pour les deux attaques sans distinction puisque l'observateur ne cherche pas à isoler l'attaque et se concentre uniquement sur les conséquences de l'attaque. Pour*recall* , étant donné que nous ne pouvons pas être exhaustifs sur la liste des attaques possibles, nous avons divisé les conséquences en trois parties : défaillance, étouffement et mise en place d'un contrôle malveillant. Ainsi, nous avons pu tester les deux premières parties dans les deux sections précédentes. On vient de voir que la défaillance provoquée par un attaquant, suite à une attaque *Control Message Drop Attack* présentée dans (Lee et al., 2017a) et dans le chapitre 1, qui a compromis le contrôleur est détectée exactement de la même manière. De plus, nous avons pu tester la mise en place d'un contrôle malveillant sur un algorithme de contrôle déterministe, suite à une attaque *Flow Rule Modification Attack* présentée dans (Lee et al., 2017a) et dans le chapitre 1, et l'attaque est détectée également. Cela implique également que si le contrôleur est compromis, mais qu'il rend son service alors l'observateur ne détectera pas l'attaque, mais sans que ça soit un souci parce que le service sur le réseau est rendu.

4.4.2 Dans le cas d'un algorithme de contrôle non déterministe

Cette fois on va analyser les performances de l'observateur lors de la détection d'anomalies sur un contrôle non déterministe. Pour cela nous allons utiliser la méthode multicritères proposée dans le chapitre 3. Nous avons pu la mettre en situation et voir l'importance des facteurs de pondération α_i permettant l'activation de l'un et l'autre des critères. Afin de bien expliciter les contributions de chacun des critères, nous proposons de les étudier individuellement.

4.4.2.1 Évaluation du critère 1 : performance des plans

Dans un premier temps, nous supposons que $\alpha_1 = 1$ et $\alpha_2 = 0$ tels que $\mathscr{L}(\mathcal{P}) = p_1$ afin d'observer l'impact du premier critère p_1 sur le calcul de vraisemblance.

Scénario

La topologie considérée ici est la topologie du réseau européen GÉANT à 23 nœuds et 37 liens. Cette topologie est rappelée en Fig. 4.18. Le trafic injecté sera celui du dataset (TOTEM, January, 2006) (également référencé par la librairie SNDlib [16]).

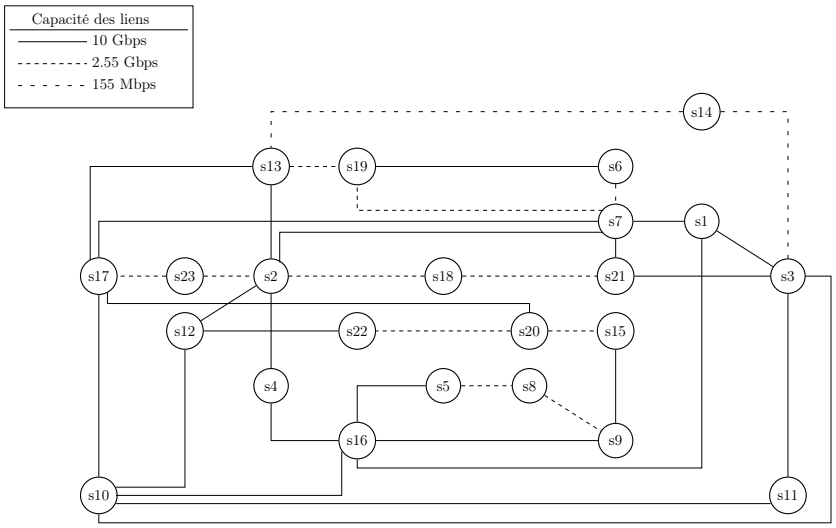


FIGURE 4.18 – Topologie du réseau GÉANT à 23 nœuds.

Le contrôle considéré est celui présenté dans (Casas-Velasco et al., 2020) et disponible depuis (Casas-Velasco, 2020). Ce contrôle est proactif, ce qui signifie que périodiquement un nouveau plan de données est mis en place. Ici, l'algorithme de contrôle est basé sur une méthode type algorithme glouton qui implique un aspect aléatoire et donc dans le même contexte le contrôleur peut mettre en place un plan de données différent. L'architecture s'appelle *Reinforcement Learning and Software-Defined Networking Intelligent Routing (RSIR)*. Ils définissent un algorithme de routage proactif basé sur l'apprentissage par renforcement (RL) qui prend en compte les informations sur l'état des liens pour explorer, apprendre et exploiter les chemins potentiels même pendant les changements dynamiques du trafic. L'objectif étant de minimiser les paramètres de qualité de service suivant : bande passante disponible, retard et perte de paquets. Plus précisément, la récompense R à maximiser par la méthode d'apprentissage par renforcement est définie dans (Casas-Velasco et al., 2020) de la façon suivante :

$$R = \beta_1 \times \frac{1}{bwa_{link}} + \beta_2 \times d_{link} + \beta_3 \times l_{link} \tag{4.1}$$

avec :
— bwa_{link} : Bande passante disponible par lien.
— d_{link} : Le délai par lien.
— l_{link} : le taux de paquet perdu par lien.
— β_i : les pondérations (arbitraire) de chaque métrique.

16. `http://sndlib.zib.de/home.action`

La structure de l'algorithme de contrôle est représentée en Fig. 4.19.

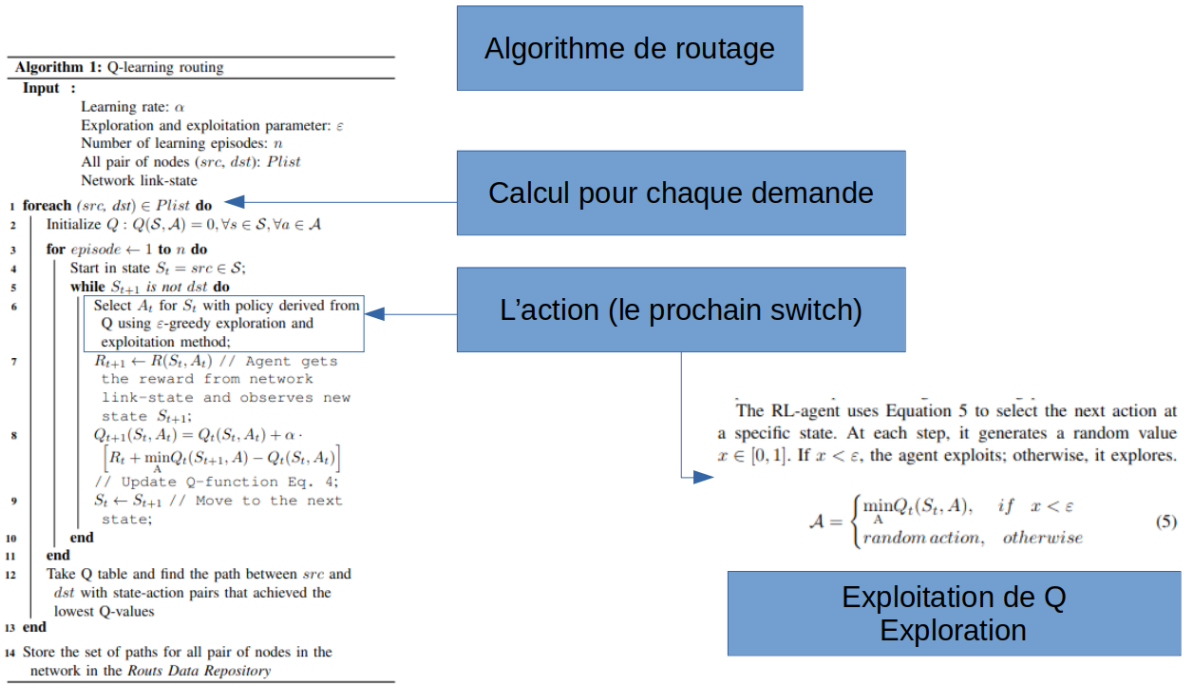


FIGURE 4.19 – Structure de la fonction de commande non déterministe étudiée (Casas-Velasco et al., 2020).

À chaque nouveau cycle, le contrôleur parcours les performances de potentiellement 506 chemins. L'objectif est d'explorer un maximum de solutions pour augmenter les chances de tomber sur l'optimum global et non juste local. La façon de construire le chemin est basée sur un algorithme glouton (greedy-algorithm). Un réel est tiré entre 0 et 1. Selon la valeur (et le coefficient de glouton, ici 0.8) le choix de la prochaine action, c'est-à-dire le prochain saut, sera différent : soit ce sera aléatoire, soit ce sera la moins "coûteuse". En effet, pour chaque action le coût (ou qualité) est calculé, dans (Casas-Velasco et al., 2020), de la manière suivante :

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha \times [R_t + min_A Q_t(S_{t+1}, A_t) - Q_t(S_t, A_t)] \tag{4.2}$$

$$Q_0(s, a) = 0 \forall s \in S \forall a \in A \tag{4.3}$$

avec :

— S_t : le nœud à l'étape t du chemin.
— A_t : l'action qui définira le prochain nœud à l'étape t.
— α : un paramètre qui définit le taux d'apprentissage, c'est-à-dire le poids de l'information nouvellement acquise par rapport à l'information précédente

En pratique, il s'agit de dire que le coût (ou qualité) d'une action est la moyenne entre le coût précédent et le coût de la prochaine action (ici la récompense du lien ajoutée au coût minimum de la prochaine action). Pour chaque demande, on calcule 300 fois le chemin afin de regarder l'ensemble des possibilités et en sélectionner la meilleure.

L'algorithme de routage est disponible dans (Casas-Velasco, 2020), est implémenté dans un contrôleur Ryu (Team, 2012).

Rappelons que l'algorithme est non-déterministe. En effet, pour une même source et une même destination, deux chemins différents peuvent être mis en place. À titre illustratif, en Fig. 4.20a et Fig. 4.20b on a représenté deux solutions différentes mises en place par le contrôleur. On peut voir que pour le flux de 10.0.0.19 en direction de 10.0.0.23, au niveau du switch 23, le chemin 1 part du côté du port *eth*4 alors que le chemin 2 part vers le port *eth*3, de l'autre côté. Il n'y avait pas de trafic injecté, donc les chemins 19-13-17-23 ou 19-7-2-23 ont le même coût : ils empruntent des liens ayant la même capacité, sans concurrence (et donc, délai) et sans perte de paquet.

```
ip,nw_src=10.0.0.23,nw_dst=10.0.0.19 actions=output:"s19-eth1"
ip,nw_src=10.0.0.19,nw_dst=10.0.0.23 actions=output:"s19-eth4"
```

(a) Chemin 1 mis en place.

```
ip,nw_src=10.0.0.23,nw_dst=10.0.0.19 actions=output:"s19-eth1"
ip,nw_src=10.0.0.19,nw_dst=10.0.0.23 actions=output:"s19-eth3"
```

(b) Chemin 2 mis en place.

FIGURE 4.20 – Différents chemins mis en place pour une même demande.

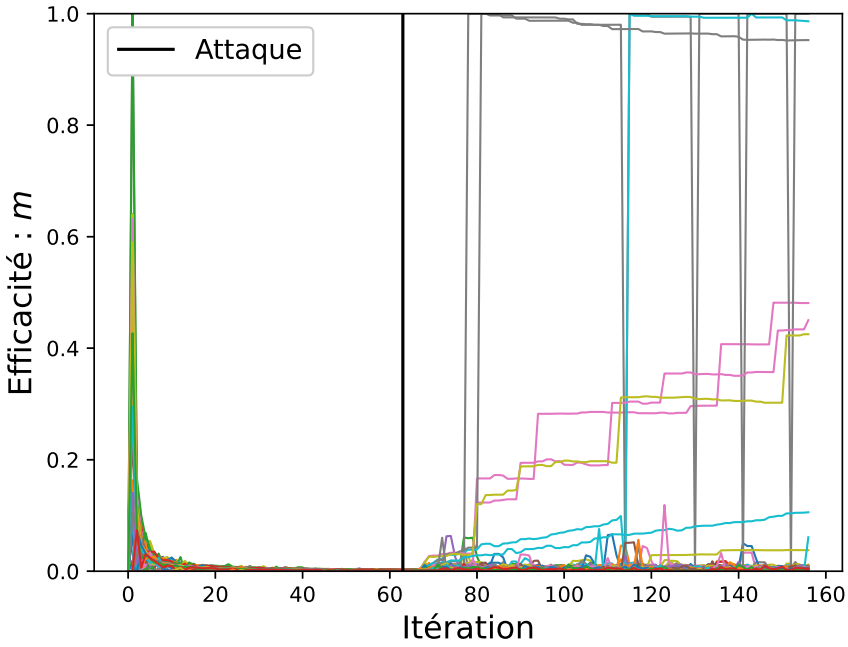
Pour mettre en place l'attaque, c'est toujours l'architecture donnée en Fig. 4.1 qui est mise en place. Le logiciel Metasploit est toujours utilisé pour compromettre le contrôleur. L'attaquant pourrait ainsi modifier les valeurs des récompenses, ou bien ajouter une application pour modifier les informations sur la topologie du réseau par exemple. Tout cela dans l'objectif de perturber le contrôle. Pour cette thèse, l'objectif de l'attaque sera de concentrer toutes les demandes sur un lien ayant la plus faible bande passante disponible. On concentrera tout sur le lien entre le switch 3 et le switch 14. Cette concentration a pour but de provoquer une saturation du réseau jusqu'à bloquer certaines communications.

Preuve de concept

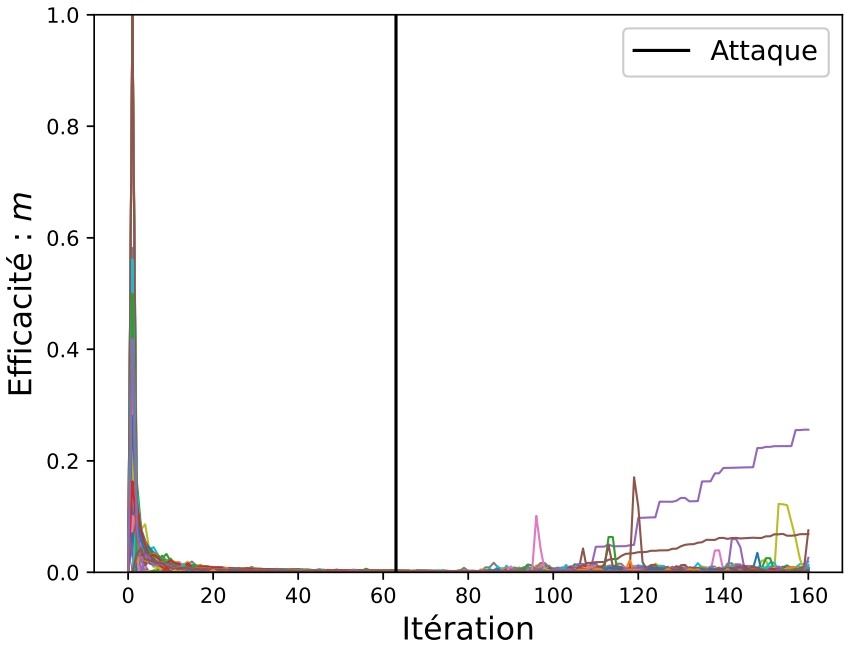
L'attaque considérée vise à mettre en place un plan de données qui conduit à une saturation dans le réseau. En fait, au lieu des routes calculées par l'algorithme RSIR, l'attaquant va établir des routes qui passent par le lien ayant la bande passante disponible la plus faible, et comme la demande totale sur le réseau est plus grande que la capacité de ce lien, une saturation se produira.

Deux cas d'attaques sont considérés : la première est brutale en changeant toutes les routes en même temps et la seconde est progressive comme dans l'attaque DDoS faible et lente, en changeant seulement dix routes à la fois. Ce type d'attaque se développe de plus en plus pour échapper à la vigilance des détecteurs (Tripathi & Hubballi, 2018) (Cambiaso et al., 2019) (de Miranda Rios et al., 2021). Cela permettra de voir la réaction de l'observateur face à ces divers types d'attaques et donc observer les limites de la méthode. Les Fig. 4.21a et Fig. 4.21b traitent de l'efficacité du plan, $m(\mathcal{P}, (s, d))$, tandis que les Fig. 4.22a et Fig. 4.22b montrent l'écart absolu entre ce qui est reçu et transmis, $|N_{Send} - N_{Received}|$, pour chaque flux. Il y a une couleur par flux et on ne détaille pas les flux puisqu'on ne les distingue pas dans le traitement.

On peut observer sur la Fig. 4.21a et la Fig. 4.21b qu'il y a des pertes au début des expériences (les 10 premières itérations). En effet, tout au début de l'expérience, il y a des paquets qui sont envoyés, mais qui, à cause de la latence naturelle du réseau, ne sont pas encore arrivés. Dans la mesure où il s'agit de l'établissement transitoire des communications (et chemins), nous ne retiendrons pas ces points dans la suite. Pour les 70 premières itérations, le plan de données calculé

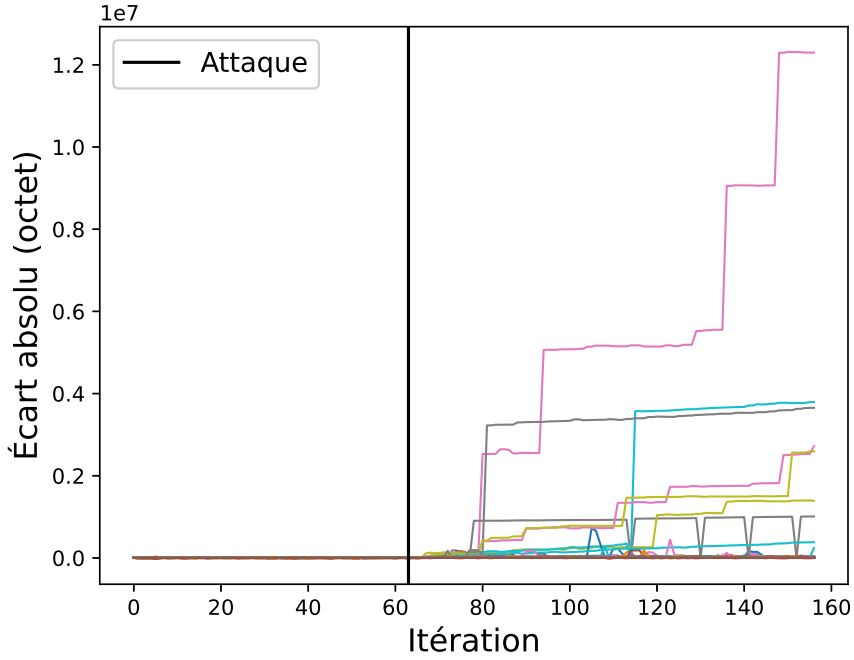


(a) Attaque brutale

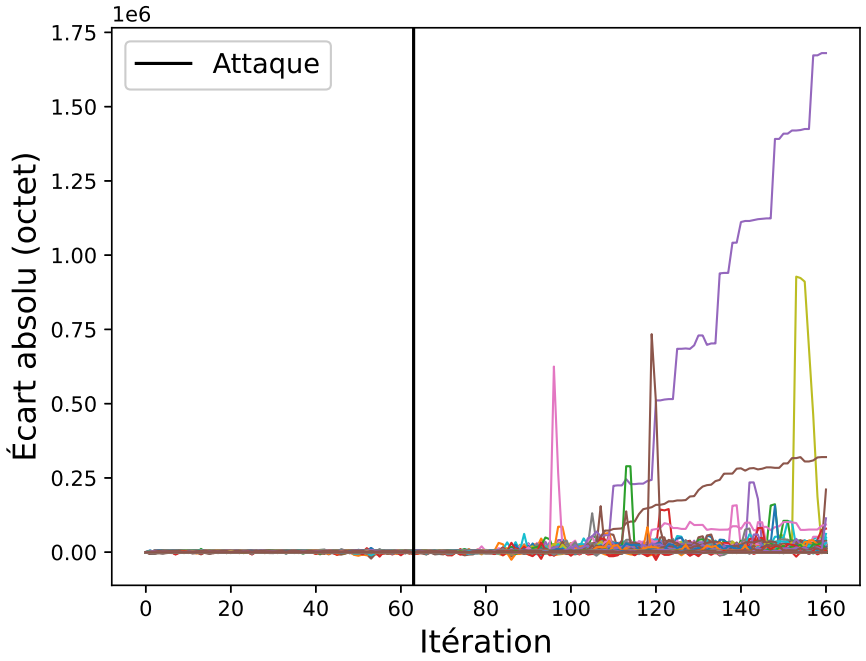


(b) Attaque douce

FIGURE 4.21 – Efficacité du plan lors des deux attaques mises en place. Une couleur correspond à une demande.



(a) Attaque brutale



(b) Attaque douce

FIGURE 4.22 – Écart absolu entre ce qui est reçu et transmis dans le cas des deux attaques mises en place. Une couleur correspond à une demande.

par le contrôleur n'est pas modifié, l'attaque commence à la 71ème itération. Pour l'attaque brutale, les effets sont visibles autour de la 80ème itération et on peut voir sur Fig. 4.21a que cela entraîne immédiatement des pertes de paquets. Tandis que l'attaque douce a un impact plus faible comme souhaité et dont la dérivée est relativement constante même si l'écart entre ce qui est envoyé et reçu augmente bel et bien. Par ailleurs, on peut constater, en comparant Fig. 4.22a et Fig. 4.22b que l'écart entre les deux évolutions en valeur absolue ($\equiv 10^7$ bytes en cas d'attaque brutale contre $\equiv 10^6$ bytes en cas d'attaque douce). Cela montre bien la différence entre les deux types d'attaques.

L'évolution du critère p_1 est finalement donnée dans la Fig. 4.23. Le seuil a été fixé à $TD = 0.8$.

On peut observer, sur Fig. 4.23b, que pour une attaque de type douce le critère n'évolue pas significativement et reste relativement constant. Ici, il reste au-dessus du seuil et aucune alarme n'est déclenchée. En revanche, en cas d'attaque brutale, en Fig. 4.23a, une alarme est déclenchée. Avec la configuration proposée, le déclenchement se fait après 20 mesures des statistiques (~ 400 secondes). Néanmoins, la réactivité et la sensibilité de la détection à l'aide de ce critère dépendent de la tolérance fixée. Par exemple ici, l'attaque douce n'est pas détectée.

La figure Fig. 4.23 met en évidence que pour $i < j$ la valeur du critère p_1 pour $depth = i$ est plus faible que pour $depth = j$. On observe que la courbe de p_1 pour $depth = j$ est en dessous de celle pour $depth = i$. Mathématiquement, en considérant les ensembles d'observations précédentes S_i avec une profondeur i et S_j avec une profondeur j (tel que $i < j$) nous donne :

$$S_{depth=i} \subset S_{depth=j} \Rightarrow max_{\mathcal{P} \in S_i} \varphi(\mathcal{P}_O, \mathcal{P}) \leq max_{\mathcal{P} \in S_j} \varphi(\mathcal{P}_O, \mathcal{P})$$

$$\Rightarrow 1 - max_{\mathcal{P} \in S_i} \varphi(\mathcal{P}_O, \mathcal{P}) \geq 1 - max_{\mathcal{P} \in S_j} \varphi(\mathcal{P}_O, \mathcal{P})$$

$$p_{1,depth=i} \geq p_{1,depth=j}$$

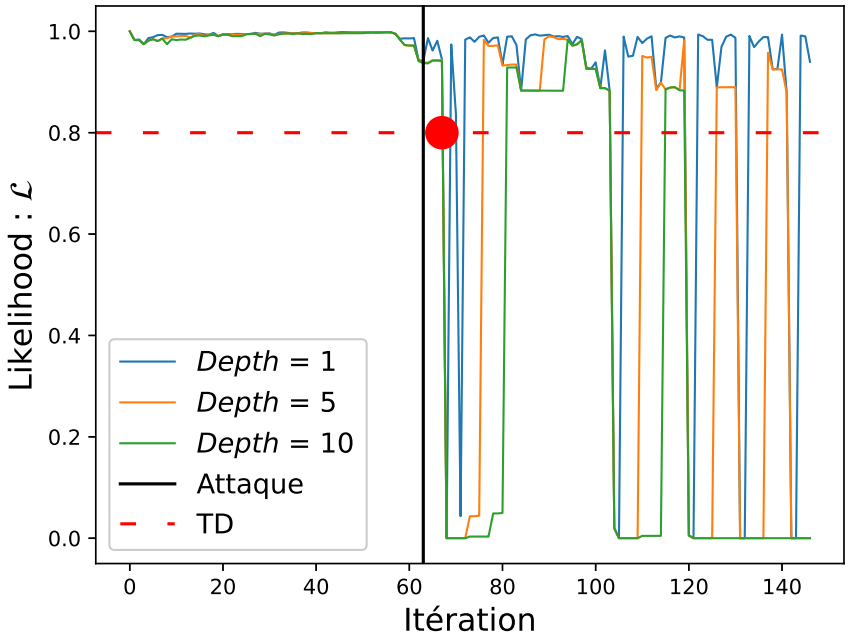
En conséquence, pour étudier les effets du seuil sur les résultats de ce critère, nous allons considérer le cas le plus évolutif c'est-à-dire $depth = 1$. Sinon on aurait des résultats constants et inhérents à un ensemble d'observations passé ce qui biaise les résultats.

Discussion sur le seuil TD

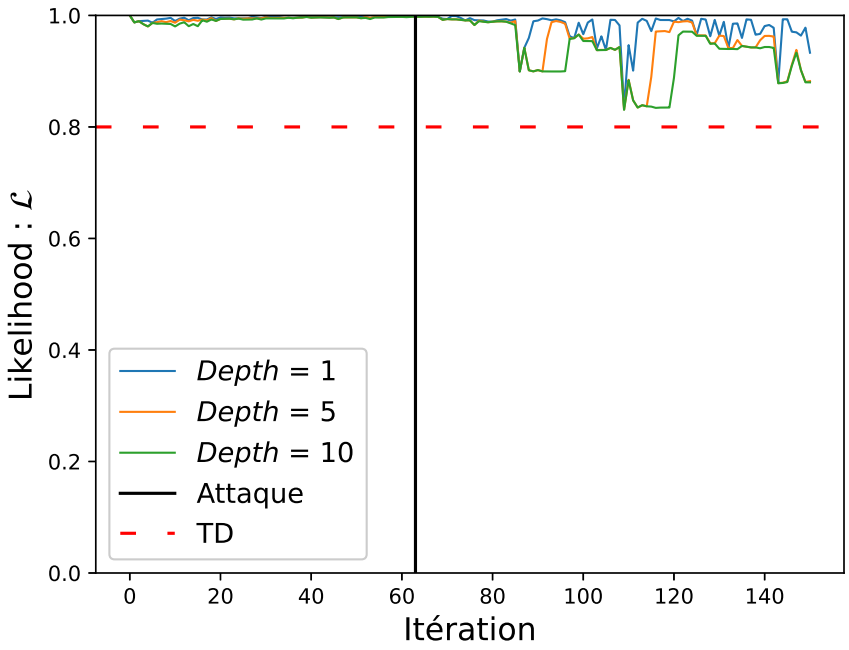
Dans cette section nous avons divisé l'expérience en trois phases :

1. Les 30 premières itérations : phase nominale. Pas d'attaque.
2. De l'itération 30 à 130 : phase d'attaque où les chemins qui vont passer par le lien de faible capacité sont choisis selon une distribution aléatoire.
3. De l'itération 130 à 160 : phase d'attaque où les 506 chemins passent par le lien de faible capacité.

Les résultats sont donnés dans la Fig. 4.24. Fig. 4.24b représente le nombre de chemins, par plan de données, passant par le lien entre le switch 3 et le switch 14. Sur Fig. 4.24a on peut retrouver l'évolution du critère p_1. De plus, nous ajoutons trois lignes horizontales qui correspondent à trois exemples de seuil TD : $0,7$, $0,8$ et $0,9$. Cette limite signifie que 10% (ou 20 30) des données transmises peuvent ne pas être encore arrivées. Chaque point correspond à une alarme, les points bleus représentent une alarme pour les 3 limites, les points rouges uniquement pour $0,8$ et $0,9$ alors que les gris c'est uniquement pour $0,9$. Ces limites dépendent des exigences du réseau, les données ne sont pas arrivées, mais peuvent arriver plus tard et donc la tolérance fixée dépend de la latence tolérée. Ici, nous allons observer l'impact du seuil concernant trois valeurs différentes.

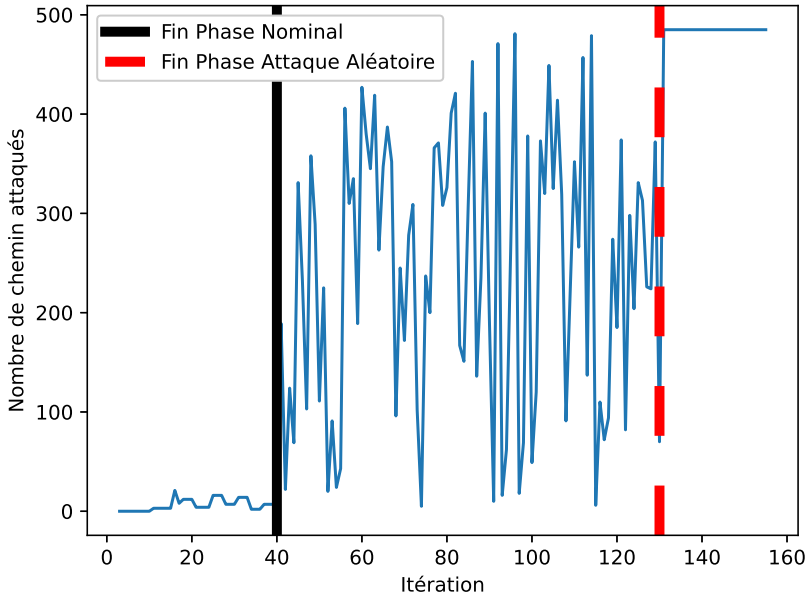


(a) Attaque brutale

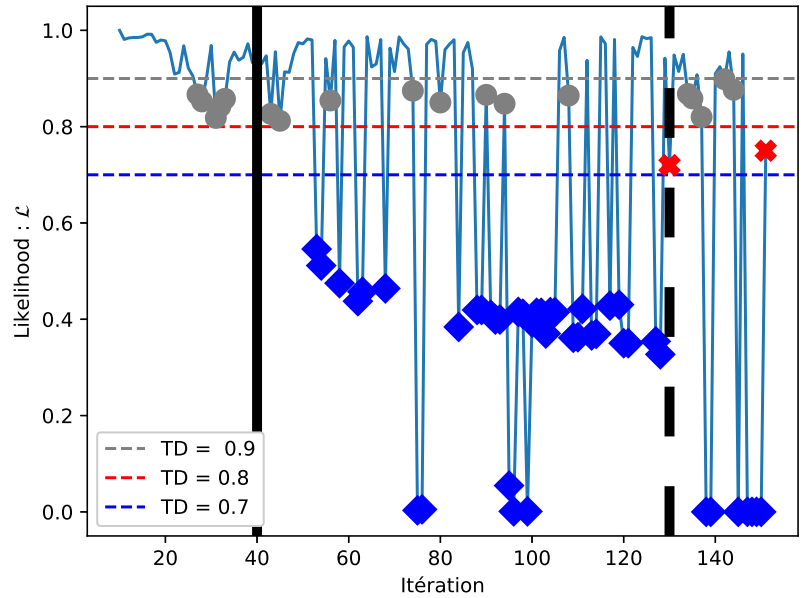


(b) Attaque douce

FIGURE 4.23 – Vraisemblance des plans selon le critère p_1 lors de la compromission du contrôleur visant à provoquer une saturation sur le réseau.



(a) Évolution de la distance



(b) Nombre de chemins passant par le lien entre le switch 3 et le switch 14 par itération.

FIGURE 4.24 – Évolution de la vraisemblance selon le critère p_1 dans le cas d'une attaque au comportement aléatoire. Il y a une phase nominale, puis une phase d'attaque au comportement aléatoire et enfin une phase d'attaque au pire comportement.

	Seuil	Précision	*Recall*	Justesse
ht	0,9	0,89	0,55	0,63
	0,8	1,0	0,44	0,60
	0,7	1,0	0,42	0,58

TABLE 4.3 – Performances du critère 1.

En ce qui concerne le seuil $TD = 0,9$, il y a évidemment plus d'alarmes que les autres seuils ($0,7$ et $0,8$) en raison du manque de tolérance. Mais, comme on peut le voir autour de la 40ième itération, une alarme est levée alors que le critère remonte au-dessus de la limite juste après. Est il alors justifié de lever une alarme aussi rapidement ? Une telle latence peut être la cause de l'apparition d'une grosse demande ou s'agit-il d'un autre événement qui n'est pas du ressort du contrôle ? Cependant, l'augmentation de la valeur du seuil et donc de la tolérance a un impact sur la réactivité. On peut observer que pour $TD = 0,7$, la première alarme est levée après 55 itérations, ce qui signifie que le service est endommagé au cours des 15 itérations précédentes. Une telle tolérance peut conduire au pire cas, comme dans la Fig. 4.23b, où aucune attaque n'est détectée.

Ainsi, le seuil correspond à un compromis entre la réactivité et la précision de la détection. Les valeurs des métriques sont données dans le tableau. 4.3.

Concentrons-nous sur la précision. On peut observer que la précision augmente avec la tolérance en raison du nombre de faux positifs comme on peut l'observer sur la Fig. 4.24a. Néanmoins, cette précision est due à un laxisme et une forte tolérance qui se répercute les autres métriques.

Considérons maintenant le*recall* . Cette propriété correspond à la comparaison du nombre d'attaques détectées par rapport au nombre d'attaques qui auraient dû être détectées. Le taux de faux négatif augmente lorsque la tolérance augmente, mais la question est de savoir comment interpréter ces faux négatifs. Le but de la méthode est de vérifier la cohérence du contrôle et donc si les performances des décisions du contrôleur sont dans la tolérance, ce n'est pas un problème, même si les performances ne sont pas optimales. Cependant, une conséquence de la prise en compte d'une tolérance élevée est d'être sensible aux attaques lentes comme présentées dans la figure. 4.23b.

En conclusion, la précision augmente lorsque la tolérance diminue, car le *recall* évolue plus significativement que la précision. Néanmoins, le choix du seuil dépend du service et de la tolérance attendus pour le réseau et représente un compromis entre la réactivité souhaitée et la qualité de service requise sur le réseau.

Par ailleurs, dans la définition proposée de p_1 nous avons mis toutes les demandes au même niveau. Il se peut que certaines demandes sur le réseau soient plus importantes que d'autres et il est donc possible d'associer un seuil différent pour chaque demande. Certaines pouvant être plus critiques que d'autres de sorte que les tolérances ne seront pas les mêmes. On peut alors les dissocier et considérer plusieurs critères différents : un par niveau d'importance de flux.

4.4.2.2 Évaluation du critère 2 : vraisemblance des routes

Dans cette partie, nous supposons que $\alpha_1 = 0$ et $\alpha_2 = 1$ tels que $\mathscr{L}(\mathcal{P}) = p_2$ l'objectif est d'observer l'impact du second critère sur le calcul de vraisemblance, c'est-à-dire la vraisemblance de la séquence observée.

Scénario

On considère toujours la topologie du réseau GÉANT à 23 nœuds et 37 liens présenté en Fig. 4.18.

Cependant cette fois nous allons modifier les solutions (plans) de routage mis en œuvre. Le but de cette section est d'étudier les performances du critère 2 correspondant à l'étude de la structure d'un plan de données observé. Les modèles proposés, au chapitre 3, sont ainsi utilisés pour faire une analyse statistique des plans observés afin d'établir le score de vraisemblance d'un plan observé. C'est la raison pour laquelle nous allons étudier les performances de ce critère selon la variance des plans. Ainsi, à partir d'un ensemble de plans mis en place par (Casas-Velasco et al., 2020), un nouveau plan sera mis en place à chaque itération après un tirage aléatoire selon une distribution de Poisson à paramètre fixe. La distribution n'a pas d'importance, les expérimentations présentées ci-après ont été testées sur la loi binomiale et les résultats étaient similaires. Le paramètre principal est la variance de la distribution. Ce paramètre définit la variabilité des plans et nous allons analyser l'impact de cette variabilité dans les résultats en fonction du modèle. Nous considérons donc un contrôle C comme suit :

$$C \sim \mathcal{P}(\lambda) \tag{4.4}$$

Préparation des modèles

Les trois modèles présentés dans le chapitre 3 sont paramétrés comme suit.

Le PFA sera construit sur la base des observations de la phase d'apprentissage sans paramétrage précis.

Pour le HMM, afin de palier les limitations de calcul, lié à la complexité des algorithmes tel Baum-Welch, nous fixerons $N = 3$ états internes.

Un résumé de la configuration choisie pour le RNN est donné dans le tableau. 4.4. Les observations, qui correspondent aux entrées, sont des plans de données et nous proposons d'associer à chaque plan de données un score correspondant à la fréquence d'apparition attendue dans ce contexte. De plus, pour le RNN, la valeur x du jeu de données D doit être normalisée dans l'intervalle [0, 1] par un calcul min-max comme suit :

$$x_{Norm} = \frac{x - min(D)}{max(D) - min(D)} \tag{4.5}$$

Variable	Paramètre
Couche d'entrée	5
Couche cachée	4, 3
Couche de sortie	2
Fonction d'activation	Tanh
Fonction de perte	Mean Squared Error
Batch Size	1
Epoch	20

TABLE 4.4 – Structure de notre modèle de type Reccurent Neural Network (RNN).

Phase d'apprentissage

Décrivons la phase d'apprentissage. Nous avons considéré un tirage de 5000 plans de données selon la distribution décrite juste au-dessus.

La structure du PFA est d'abord déterminée comme dans (Fouquet et al., 2020). C'est-à-dire que pour chaque observation, un état est créé en plus de l'état initial. Les transitions vont de tous les états à tous les états. Ensuite, la probabilité de chacune de ses transitions, δ, est calculée comme dans (Fouquet et al., 2020) puisque nous supposons que notre PFA est déterministe. Ainsi, pour déterminer la probabilité, $P(\delta)$, de chaque transition, nous allons compter le nombre, $N(\delta)$, de fois où le front est tiré, soit :

$$P(\delta) = \frac{N(\delta)}{\Sigma_{\delta' \in \Sigma_A | \delta'[1]=\delta[1]} N(\delta')} \tag{4.6}$$

En ce qui concerne l'emplacement initial, nous supposons qu'il existe une probabilité égale pour chaque événement.

Pour les HMM, l'algorithme de Baum-Welch est utilisé pour déterminer les paramètres du HMM.

Pour le RNN, le modèle est formé selon les paramètres donnés dans le tableau. 4.4.

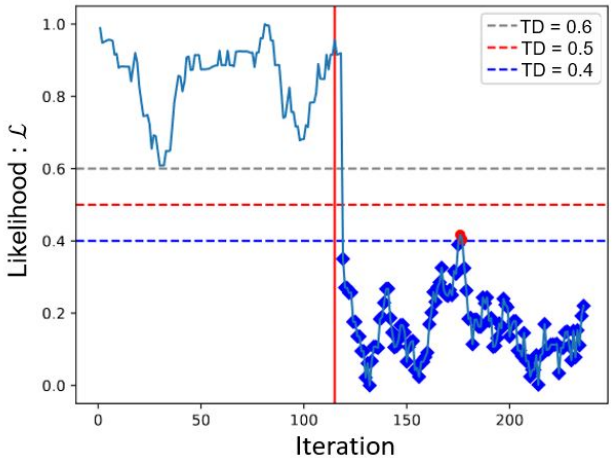
Préparation des jeux de données

Maintenant, on tire une séquence de 125 plans de données en suivant le même principe pour constituer la séquence nominale. Pour la séquence d'attaque, un tirage de 125 plans de données est effectué en utilisant une distribution uniforme. L'objectif est de mettre en place une distribution différente pendant la phase nominale et la phase d'attaque. Ainsi, le processus consiste en l'observation d'un plan et la détermination de son score de vraisemblance afin de déterminer s'il est cohérent ou non.

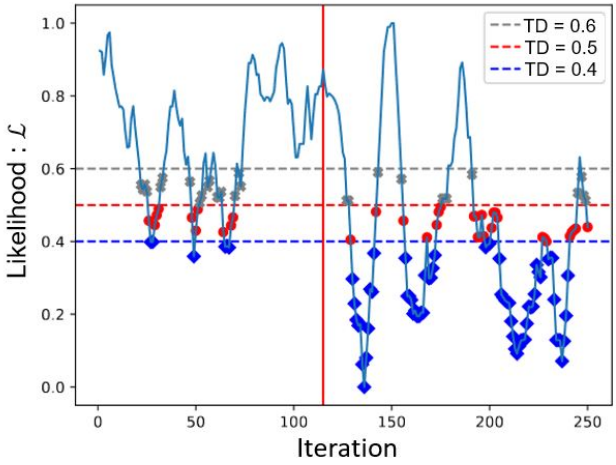
Les expériences, pour cette sous-section, ont été réalisées pour trois variances différentes : $C \sim \mathcal{P}(1)$, $C \sim \mathcal{P}(5)$ et $C \sim \mathcal{P}(15)$. Le résultat des expériences est donné dans la Fig. 4.25. Trois seuils ont été appliqués : $TD = 0,4$, $0,5$ et $0,6$. Les courbes sont divisées en deux parties : à gauche de la ligne verticale se trouve le cas nominal, sans attaque, tandis qu'à droite se trouve le cas anormal. La ligne verticale est donc à l'itération numéro 125. Les alarmes sont représentées par un point particulier (en fonction du seuil qui déclenche l'alarme : bleu en dessous lorsque l'alarme est déclenchée pour les trois seuils, rouge pour seulement $0,5$ et $0,6$ et enfin gris pour $0,6$ uniquement) sur la Fig. 4.25.

Tout d'abord, on peut observer pour le PFA et le HMM que la distinction entre l'attaque et le cas nominal est moins claire dès que la variance augmente. En effet, le but de la méthode est de détecter toute incohérence dans le contrôle, mais lorsque la variance augmente, la différence entre les deux distributions n'est pas claire. Ainsi, le PFA et le HMM tendent à confondre les deux distributions. En ce qui concerne le PFA, le décalage entre la vraisemblance dans un cas d'attaque ou dans un cas nominal diminue avec l'augmentation de la variance : le décalage est clair pour la Fig. 4.25a, autour de $0,8$ dans le cas nominal au lieu de $0,3$ pendant l'attaque. Mais, quel que soit le cas, les courbes tendent vers $0,5$ ce qui signifie que l'observateur tend à l'indécision et n'est pas capable de dissocier l'attaque du comportement nominal comme on peut l'observer sur la Fig. 4.25c.

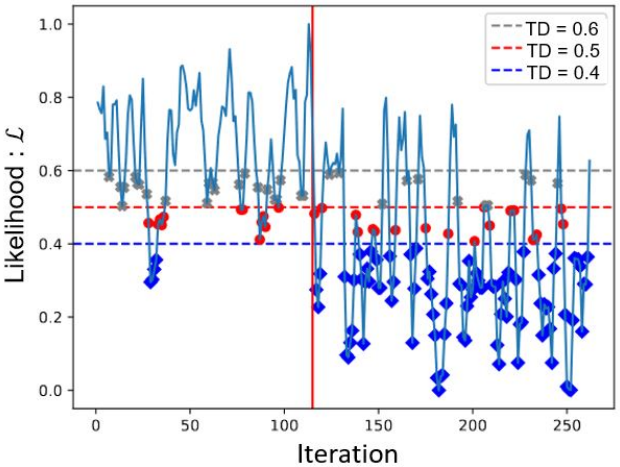
Il en va de même pour le score de vraisemblance déterminé à l'aide des HMM comme on peut le voir sur la Fig. 4.26a, autour de $0,9$ pendant le cas nominal et de $0,4$ pendant l'attaque. Mais,



(a) PFA : Variance = 1

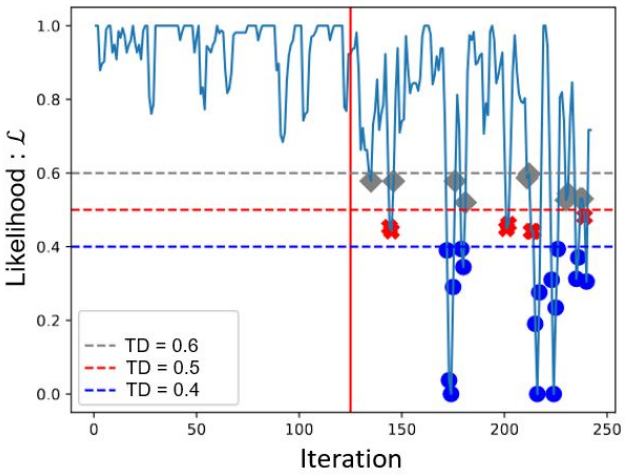


(b) PFA : Variance = 5

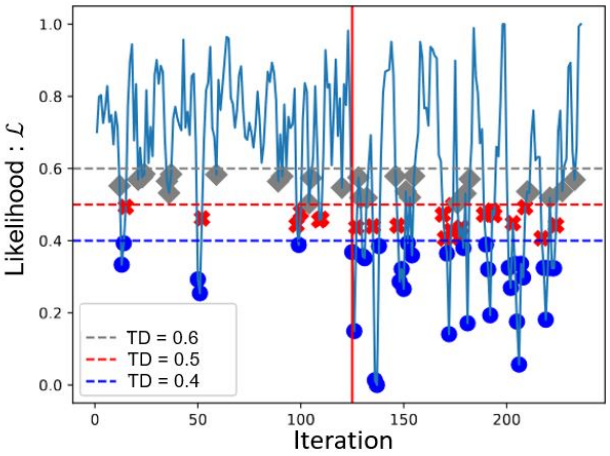


(c) PFA : Variance = 15

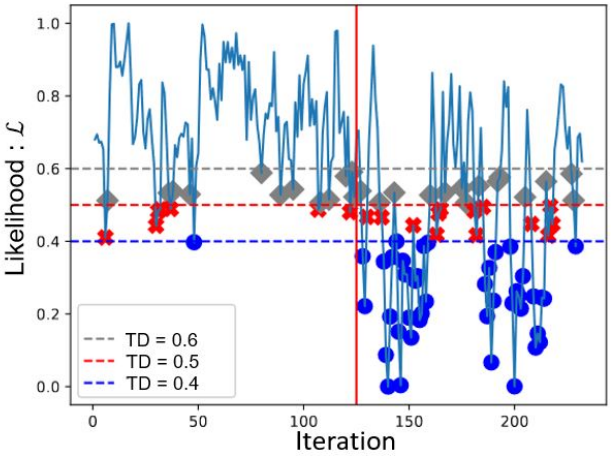
FIGURE 4.25 – Comparaison de la vraisemblance normalisée d'une séquence nominale (125 premières itérations) comparée à une séquence attaquée (125 dernières) en utilisant le modèle PFA.



(a) HMM : Variance = 1

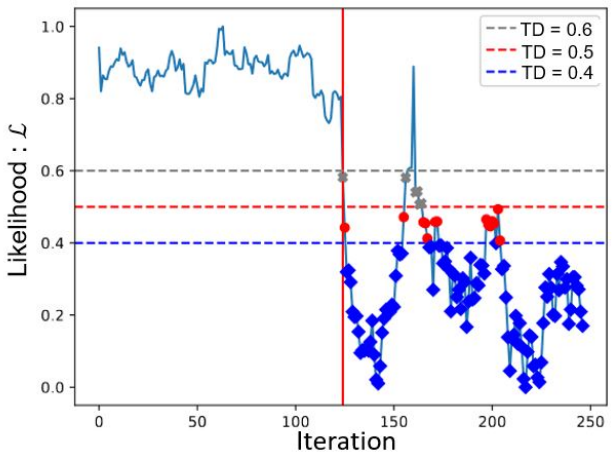


(b) HMM : Variance = 5

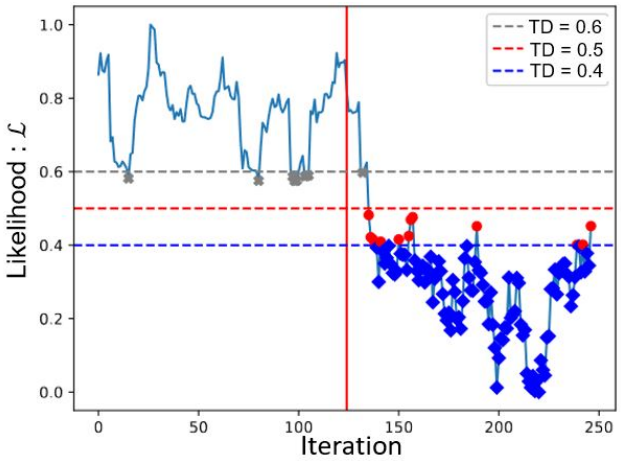


(c) HMM : Variance = 15

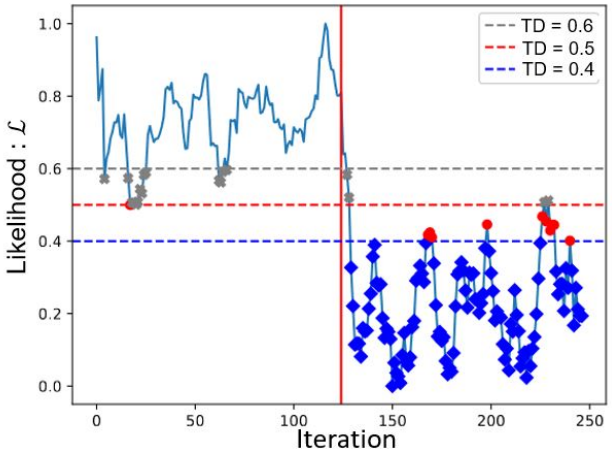
FIGURE 4.26 – Comparaison de la vraisemblance normalisée d'une séquence nominale (125 premières itérations) comparée à une séquence attaquée (125 dernières) en utilisant le modèle HMM.



(a) RNN : Variance = 1



(b) RNN : Variance = 5



(c) RNN : Variance = 15

FIGURE 4.27 – Comparaison de la vraisemblance normalisée d'une séquence nominale (125 premières itérations) comparée à une séquence attaquée (125 dernières) en utilisant le modèle RNN.

sur la Fig. 4.26c, le décalage est plus petit : environ $0,7$ pendant le cas nominal et $0,4$ pendant l'attaque.

Cependant, ce phénomène n'est pas observé avec l'approche RNN. Ceci est dû à la puissance de calcul du RNN par rapport aux deux autres, le RNN est moins sensible à l'augmentation de la variance. De plus, comme l'algorithme non déterministe est considéré, il est possible d'observer des séquences inhabituelles, mais parfaitement cohérentes et c'est la raison pour laquelle nous considérons des séquences de plans et non un seul. Bien que, plus la séquence est longue, plus notre probabilité sera précise. Mais, en ce qui concerne les HMM, la longueur de la séquence est limitée en raison de la complexité temporelle (qui a également un impact sur la réactivité de l'observateur, puisque liée à son temps de calcul), ce qui explique la différence de résultat entre les HMM et les RNN.

Pour chacun de ces modèles, nous avons testé trois seuils et on peut observer que le choix du seuil a un impact sur les résultats de la même manière que pour le critère 1. Cet impact est détaillé ci-après.

Discussion sur le seuil TD

Dans cette partie, nous allons voir l'impact du choix du seuil TD sur les résultats. Nous ne considérerons qu'un seul modèle, le PFA, et une seule variance pour la distribution du contrôle, $V = 5$. Il a été vérifié que les résultats sont similaires pour HMM, RNN et les autres variances.

Le seuil TD a été déterminé expérimentalement par rapport au pire cas L_{WC} observé. Nous proposons de fixer le seuil comme une proportion de ce pire cas, $TD = L_{WC} \times \beta$. Ainsi, nous proposons d'évaluer les métriques pour la valeur de β donné dans le tableau. 4.28.

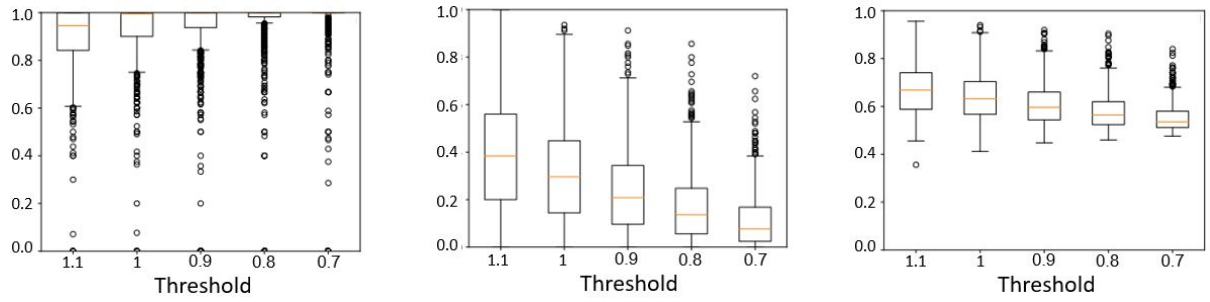
Contrôle	Observateur	
Variance	Modèle	β
5	PFA	$1,1$
		1
		$0,9$
		$0,8$
		$0,7$

FIGURE 4.28 – Les seuils utilisés

Les résultats sont représentés dans la Fig. 4.29. À première vue, on peut constater que le *recall* a augmenté alors que la précision a diminué. En effet, un compromis doit être trouvé. Concentrons-nous sur la précision. Il est clair que la précision augmente avec la limitation due au nombre de faux positifs comme on peut l'observer en comparant la Fig. 4.25. Cela signifie que la tolérance est trop stricte avec $\beta = 1,1$ par rapport à $\beta = 0,7$. Ainsi, le fait de considérer $\beta = 1,1$ est sensible, par rapport aux autres β, à un plan de données inhabituel établi par le contrôleur, ce qui est plausible en ce qui concerne un algorithme de contrôle non déterministe. Par conséquent, la diminution de β entraîne une diminution du taux de faux positifs, car l'observateur est plus tolérant. En fait, diminuer la valeur β implique d'augmenter la tolérance sur les plans de données inhabituels et donc sur les attaques possibles qui ne sont pas détectées jusqu'à ce que ces plans de données d'attaque inhabituels soient plus fréquents dans la séquence que ce qu'on est prêt à tolérer. Finalement, le risque est le même que celui décrit plus haut pour la borne temporelle.

Considérons maintenant le*recall* . Cette métrique correspond au nombre d'attaques détectées par rapport au nombre d'attaques qui auraient dû être détectées. Comme développé juste au-

dessus, le taux de vrais positifs augmente alors que β augmente. Pour des raisons similaires, le taux de faux négatifs diminue lorsque β augmente. De plus, le *recall* est inversement proportionnel à β, comme observé sur la Fig. 4.25. Enfin, on peut conclure que la tolérance a un impact sur les performances de la méthode de détection et que considérer une valeur plus grande de β entraîne une diminution du nombre d'attaques détectées, mais réduit le nombre de faux positifs (c'est-à-dire de fausses alarmes).

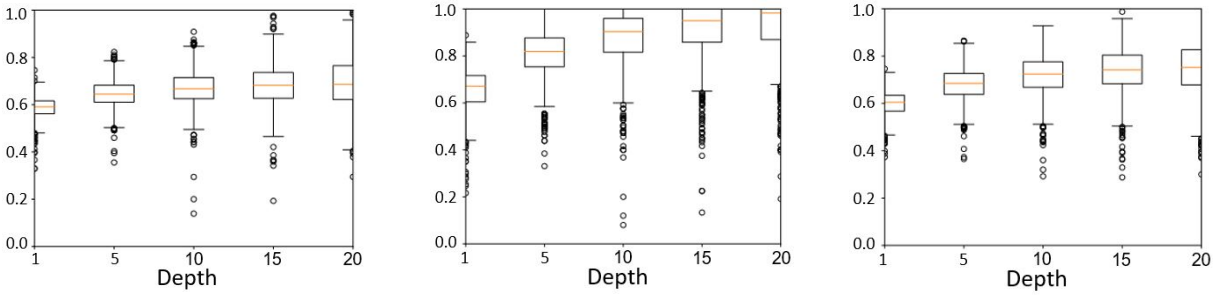


(a) Précision (b) Rappel (c) Justesse

FIGURE 4.29 – Résultats du critère p_2 selon le seuil (i.e. β).

Discussion sur la profondeur de la séquence *depth*

Il est clair que la profondeur *depth* de la séquence W utilisée pour déterminer le score de vraisemblance a un impact sur le résultat. Par exemple, dans le cas de $depth = 1$ chaque plan est plausible dans la mesure où il a déjà été observé. En effet, même si c'est rare, c'est possible dans un contexte non déterministe. Cela signifie qu'il n'y a pas de problème à l'observer à nouveau une fois, il est donc important de considérer des séquences plus profondes. Pour évaluer l'impact de la profondeur *depth* de la séquence sur les résultats, nous ne considérerons qu'un seul modèle, le PFA, et une seule variance pour la distribution du contrôle $V = 5$. Il a été vérifié que les résultats sont similaires pour HMM, RNN et les autres variances. La métrique a été évaluée pour $depth = 1, 5, 10, 15, 20$ et les résultats de la métrique en fonction de la profondeur *depth* de la séquence sont donnés dans la Fig. 4.30. Le seuil utilisé est $TD = 0,5$.



(a) Précision (b) Rappel (c) Justesse

FIGURE 4.30 – Résultat du critère p_2 selon la profondeur *depth* de la séquence analysée.

Le fait de considérer une plus grande profondeur rend l'analyse plus précise. Même si le plan observé est cohérent et fait partie des plus fréquemment observés, la séquence ne correspond pas forcément à la distribution attendue. Augmenter la profondeur *depth* de la séquence permet

de réduire la proportion des séquences plausibles dans l'ensemble des séquences observables. En conséquence, la distinction entre la séquence nominale et la séquence anormale est plus claire à mesure que la profondeur augmente, ce qui conduit à réduire le nombre de faux positifs et de faux négatifs. Ainsi, plus la profondeur de la séquence considérée par l'observateur est grande, plus il sera compliqué pour un attaquant de mettre en place une séquence de plan de données qui pourrait endommager le réseau sans être considérée comme incohérente. Enfin, une séquence plus profonde permet de distinguer les séquences anormales des séquences nominales. Cependant, considérer une valeur élevée de la profondeur a un impact sur la tolérance aux décisions inhabituelles du contrôleur, ce qui explique l'évolution de la disparité de la précision sur la Fig. 4.30a. Ceci est dû au fait que la prise en compte d'une séquence plus profonde a un impact sur le temps de calcul, et ce, dès lors que les calculs sont multipliés dès que la *depth* augmente. En ce qui concerne le PFA et le RNN, la complexité du temps de calcul est de $O(profondeur)$ alors que pour le HMM, elle est de $O(N^{profondeur})$. Par conséquent, HMM est sensible à une augmentation de la profondeur en termes de réactivité alors que ce n'est pas le cas pour PFA et RNN.

Pour vérifier ce phénomène, étudions la fonction f qui évalue la répartition entre les séquences nominales et celles attaquées définies de la façon suivante :

$$f(\Sigma, depth) = \frac{|L_{Ctrl}^{depth}|}{|L_{Att}^{depth}|} = \frac{|L_{Ctrl}^{depth}|}{|\Sigma^{depth}| - |L_{Ctrl}^{depth}|} \tag{4.7}$$

avec :
— Σ : l'ensemble des paquets OpenFlow.
— $depth$: la profondeur de séquence considérée.
— Σ^{depth} : l'ensemble des séquences de profondeur $depth$.
— L_{Ctrl}^{depth} : l'ensemble des séquences de profondeur $depth$ évaluée comme vraisemblable.
— L_{Att}^{depth} : l'ensemble des séquences de profondeur $depth$ évaluée comme non vraisemblable.
— $|X|$: le nombre d'éléments d'un ensemble X.

Ici, $|\Sigma^{depth}| = |\Sigma|^{depth}$ et $|L_{Ctrl}^{depth}|$ est déterminé expérimentalement. D'après notre expérience, l'évolution de la fonction f est donnée en Fig. 4.31 pour des profondeurs $depth > 1$.

Comme déjà évoqué, pour $depth = 1$, $\Sigma^1 = L_{Ctrl}^1$ signifie que tous les plans observés sont considérés comme vraisemblables dès lors qu'ils ont déjà été observés.

Ensuite, l'augmentation de la longueur de la séquence permet de réduire la proportion des séquences vraisemblables dans l'ensemble des séquences observables comme on peut l'observer dans la diminution des valeurs sur la Fig. 4.31. Ainsi, plus la profondeur de la séquence considérée par l'observateur est grande, plus il sera compliqué pour un attaquant de mettre en place une séquence de plan de données qui pourrait endommager le réseau sans être considérée comme non vraisemblable.

Enfin, de manière générale, la profondeur doit être fixée en prenant en compte les considérations physiques du réseau. Par exemple, selon les cas il est possible que fixer $depth = 10$, ce qui implique la prise en compte uniquement des 10 dernières observations, ne prennent pas en compte la dynamique du réseau (ex : augmentation des demandes).

Comparaison des trois solutions d'apprentissage proposées

Les expériences sont exécutées 1000 fois, ce qui permet d'avoir une bonne représentativité, et les diagrammes à moustache de la précision, du *recall* et de la justesse sont donnés respectivement dans les Fig. 4.32, Fig. 4.33 et Fig. 4.34. Le seuil utilisé est $TD = 0.5$.

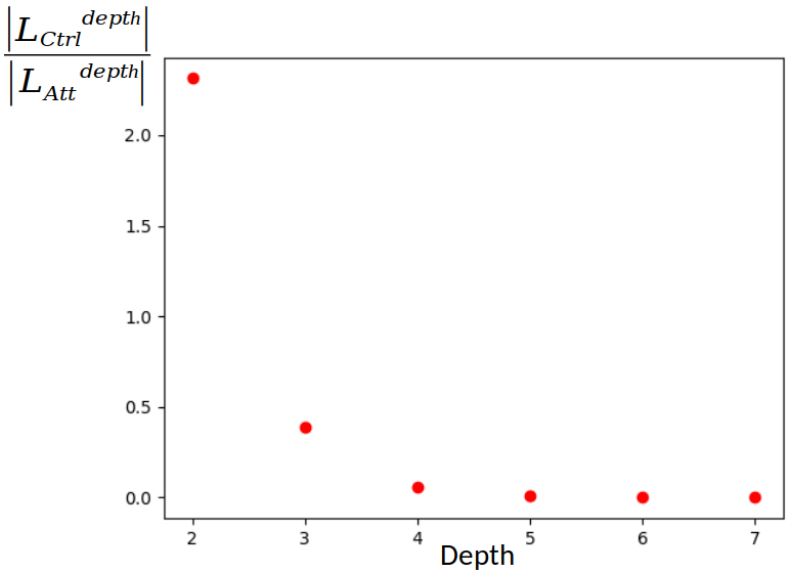


FIGURE 4.31 – Évolution de la fonction f selon la profondeur *depth*.

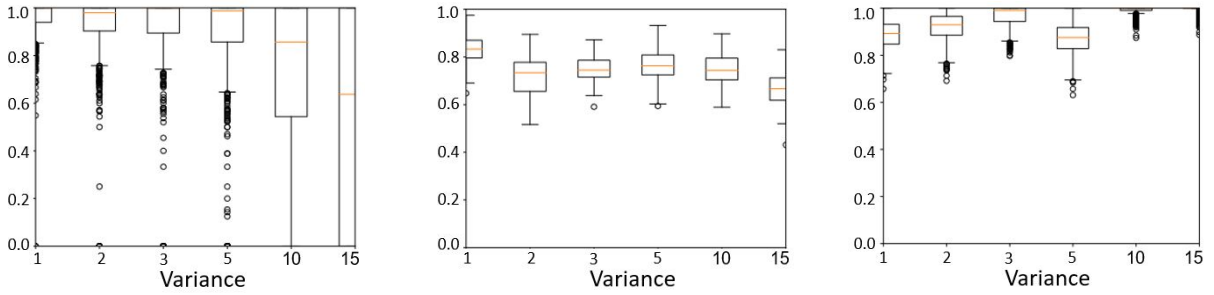
La profondeur utilisée est de 3 pour HMM (en raison de la limitation du temps de calcul comme présenté dans la section précédente) et de 10 pour PFA et RNN.

En ce qui concerne la précision, il est clair que la variance a un impact sur le modèle PFA, mais pas pour les HMM et RNN. En effet, l'augmentation de la variance a un impact sur le modèle en harmonisant toutes les probabilités de transitions et donc le PFA ne peut pas distinguer une séquence inhabituelle d'une séquence habituelle. Ceci est dû au fait que l'écart entre l'événement habituel et les événements inhabituels diminue significativement dès que la variance de la distribution augmente. Ensuite, l'évolution de la vraisemblance normalisée a diminué comme on peut le voir en comparant les Fig. 4.25a et Fig. 4.25c. Néanmoins, cet impact peut être limité en augmentant la profondeur de la séquence considérée comme présentée dans la section précédente. De plus, l'augmentation de cette profondeur *depth* est limitée à certaines considérations physiques liées à l'état du réseau.

Par ailleurs, HMM et RNN ont une précision constante sur la variance même si le RNN a de meilleures performances. Cette différence est due aux approches du modèle qui a un impact sur la puissance de calcul pour le RNN et le HMM. En raison de l'approche discrète proposée par les HMM, le nombre d'états internes est limité par la complexité de calcul des algorithmes utilisés (la complexité de l'algorithme Forward utilisé pour déterminer la vraisemblance est de $O(N^T)$ avec N le nombre d'états et T la profondeur de la séquence d'observation) alors que l'approche continue des RNN permet de traiter un plus grand nombre d'états internes. Le risque avec les RNN est alors le surapprentissage.

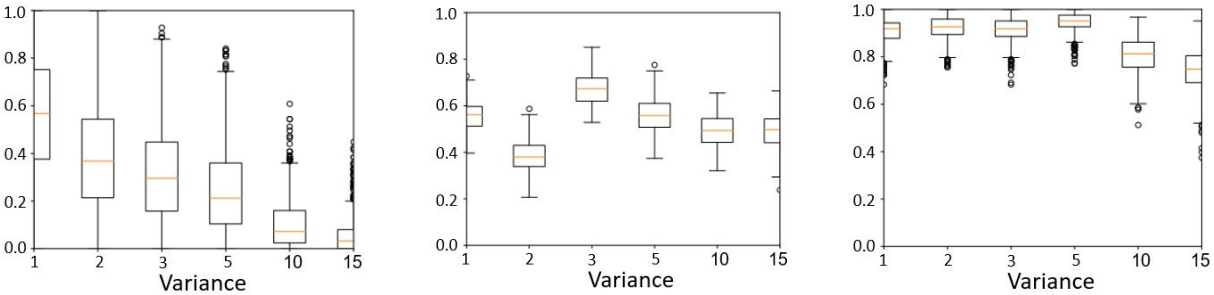
Cependant, en ce qui concerne le *recall*, il y a une différence significative entre HMM et RNN. La difficulté pour les HMM est que la profondeur de la séquence est limitée et ici nous avons fixé $depth = 3$. Cela signifie que le modèle ne prend en compte que la vraisemblance de la dernière décision de 3. Par conséquent, la distinction entre une séquence attaquée et une séquence nominale est plus fine en considérant seulement $depth = 3$ au lieu de $depth = 10$ pour le RNN. Ceci explique le faible résultat pour HMM.

En conséquence, la précision de PFA diminue lorsque la variance augmente, comme expliqué ci-dessus. Cet effet pourrait être réduit en augmentant la valeur de la profondeur de la séquence,



(a) PFA (b) HMM (c) RNN

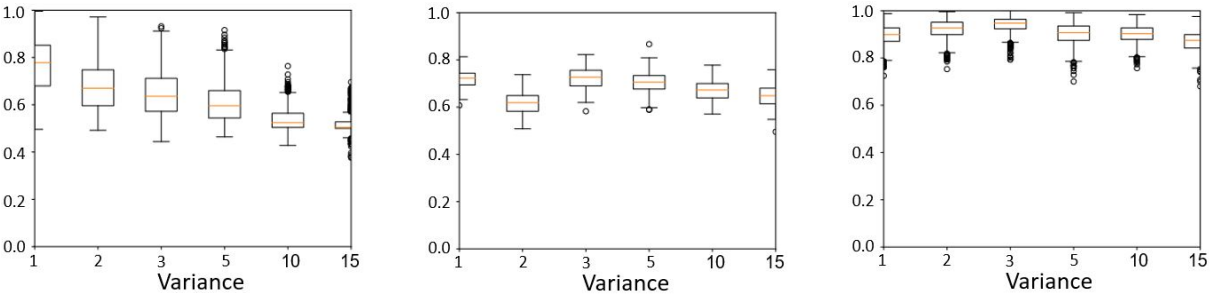
FIGURE 4.32 – Précision du critère p_2 selon la solution d'apprentissage.



(a) PFA (b) HMM (c) RNN

FIGURE 4.33 – *Recall* du critère 2 selon la solution d'apprentissage.

mais en tenant toujours compte de la dynamique du réseau. De même, la précision des HMM et RNN n'est pas sensible à l'évolution de la variance. Il existe des différences dues à l'écart entre le *recall* pour un HMM et celui pour un RNN. En fait, les résultats pendant la phase nominale sont similaires pour les HMM et les RNN, mais pas pendant la phase d'attaque en raison de la limitation de la profondeur pour les HMM comme décrite ci-dessus.



(a) PFA (b) HMM (c) RNN

FIGURE 4.34 – Justesse du critère 2 selon la solution d'apprentissage.

Pour conclure, chaque modèle a ses avantages et ses faiblesses. Le choix dépend en réalité du contrôle considéré. Nous avons proposé de diviser les algorithmes non déterministes par la variance de leur distribution. Dans le cas d'un contrôleur qui ne varie pas beaucoup dans ses décisions (variance de la distribution < 5), le PFA est bien désigné pour détecter une anomalie dans le contrôle. Cependant, ce modèle est sensible à la valeur la plus élevée de la variance du

fait que la profondeur ne peut pas être augmentée indéfiniment. HMM et RNN sont stables par rapport à la variance même si RNN est plus précis et plus exact. La principale différence entre ces deux modèles concerne les approches : discrète ou continue. Le RNN permet de multiplier les états internes alors que la puissance de calcul des HMM est limitée. Cependant, cette multiplication permet d'être plus précis, mais il y a un risque de surapprentissage comme décrit ci-dessus. Ainsi, le modèle final doit être un compromis entre la précision nécessaire en fonction de la variance du contrôleur. Par ailleurs, les solutions proposées ne sont pas optimales puisque ce n'est pas le but de ces travaux. Nous avons discuté sur l'apport de chaque niveau de modélisation pour notre problème de détection, mais d'autres modèles auraient pu être utilisés. En particulier, sur un cas d'étude, une étude comparative entre les différents algorithmes d'apprentissages classiques sera menée afin de déterminer le meilleur modèle propre à ce cas d'étude.

Un autre point est que nous considérons que le contrôleur suit une distribution unique, mais il est possible que les décisions suivent une distribution différente selon le contexte. Si c'est le cas, alors un seul PFA ou un seul HMM n'est pas suffisant : il en faut un par distribution (et donc par contexte) alors que les RNN peuvent gérer de telles distributions.

4.4.2.3 Limite des solutions proposées : évolution de la distribution

Nous venons d'évaluer les performances de l'observateur selon la variance d'un contrôle qui suit une loi de Poisson C telle que $C \sim \mathcal{P}(\lambda)$. Or, cette distribution peut être amenée à évoluer selon les contextes. Par exemple, considérons le trafic du jeu de données TOTEM [17] (également référencé par la librairie SNDlib [18]) (TOTEM, January, 2006), les évolutions sont représentées en Fig. 3.8.

On peut voir que la demande n'est pas uniforme. Que ça soit la demande totale qui est cyclique ou même chaque demande qui suit le même comportement avec certaines augmentations soudaines. En conséquence, la distribution suivie par le contrôleur peut évoluer selon le type de contrôle. Auquel cas, il y a deux façons de construire l'observateur : un modèle par phase du cycle ou un seul modèle global. Dans le cas d'un modèle par cycle, nous avons étudié l'apport de chacun des modèles précédemment. Dans le cas d'un seul modèle global, les modèles PFA et HMM ne sont plus utilisables, soit compte tenu des limites théoriques ou par des limites de puissance de calcul. En effet, les modèles pertinents sont ceux de machine learning supervisée tels que SVM, kNN ou RNN.

D'après Fig. 3.8, le cycle de l'évolution du trafic peut être divisé en quatre parties : le passage par le minimum local des données injectées, phase de croissance, passage par le maximum local et enfin la phase de décroissance. Nous mettons en place un contrôleur avec une variance associée à chaque phase. En associant à chaque plan de données un identifiant, nous avons représenté une séquence de contrôle cyclique, où chaque cycle est divisé en quatre parties (voir sur la Fig. 4.35). À gauche de la ligne verticale il s'agit de la phase nominale où un cycle est divisé en quatre distributions différentes centrées sur respectivement : 4, 50, 10 et 100. Alors qu'à droite, il s'agit de la phase d'attaque. Pour modéliser la phase d'attaque, nous avons également mis un cycle divisé en trois phases : deux suivant une loi de Poisson centrée sur 15 et 150 et une troisième suivant une loi uniforme.

Les résultats de la détection en utilisant kNN, SVM et RNN sont donnés en Fig. 4.36. On n'a considéré qu'une profondeur $depth = 10$. On peut voir que ces trois modèles permettent de détecter l'anomalie. Par ailleurs, les analyses précédentes sur l'impact de la borne ainsi que

17. `https://totem.info.ucl.ac.be/dataset.html`
18. `http://sndlib.zib.de/home.action`

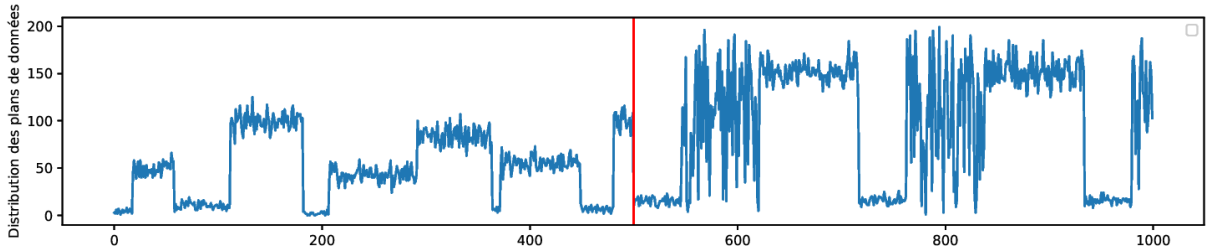
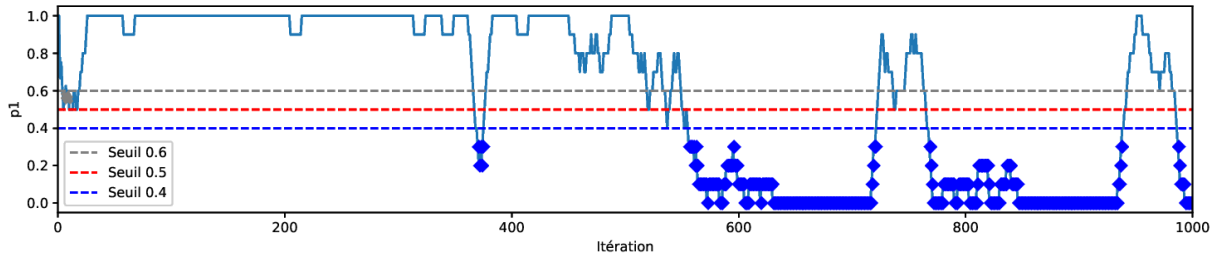
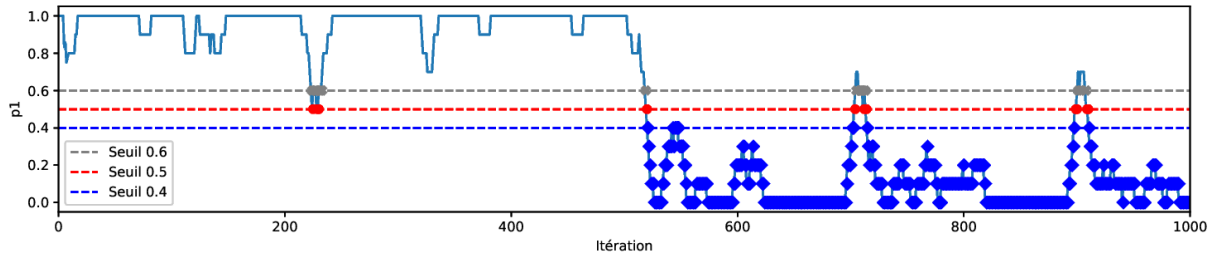


FIGURE 4.35 – Évolution des choix du contrôleur en phase nominale (les 500 premières itérations) et en phase d'attaque (les 500 dernières itérations) dans le cas d'un contrôleur suivant différentes distributions.

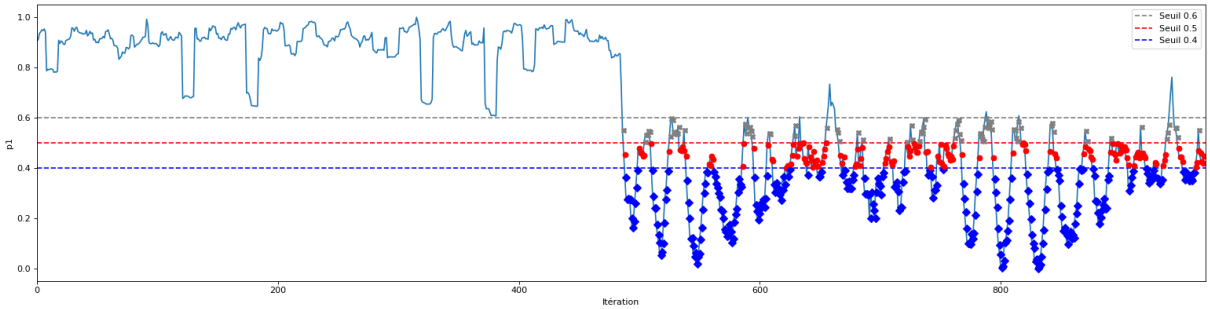
la profondeur de la séquence sont toujours valables, mais ne seront pas plus détaillées ici. De plus, il est possible de développer les analyses afin de déterminer lequel de ces modèles offre les meilleures performances, mais cette optimalité est propre au cas d'étude considéré, ne peut être généralisé et on ne fera donc pas de commentaire plus approfondi.



(a) SVM



(b) kNN



(c) RNN

FIGURE 4.36 – Résultats de la détection de l'observateur face à un contrôleur suivant plusieurs distributions en utilisant SVM, kNN et RNN.

4.5 Conclusion

En conclusion, le premier constat à établir est que la détection depuis l'observateur, sans échange direct avec le contrôleur, fonctionne. En outre, cela valide les propositions faites dans les chapitres précédents. L'observateur réussit à détecter des attaques et des défaillances du contrôleur de la même façon, sans distinction et sans isolation. Nous avons confronté l'observateur à différents scénarios et différentes attaques visant le contrôleur. On a pu voir que ces attaques sont détectées dès lors que l'attaque se répercute sur l'activité de la commande. Sans cela, l'observateur ne détecte rien, mais ce qui n'est pas un problème puisque le service est toujours rendu par le contrôleur Cependant, bien que la détection soit faite, on peut noter la difficulté de prise de décision avec une certaine proportion de mauvaise décision. Cette proportion dépend des paramètres qui ont été fixés. Nous avons pu discuter de l'influence des différents paramètres et seuils en isolant leurs comportements. Enfin, on notera toutefois qu'il reste compliqué d'en ressortir une préconisation précise attendu que celle-ci est dépendante du cas d'étude et des contraintes de sécurité à mettre en œuvre.

Conclusions et perspectives

En conclusion, ces travaux ont présenté une méthode de détection d'anomalie dans un contrôle de réseau. Pour ce faire, les bénéfices de l'architecture centralisée Software-Defined Networking ont permis de mettre en application de cette méthode de détection. Ce type d'architecture se singularise par l'introduction d'un contrôleur de réseau qui est le seul responsable du contrôle pour l'ensemble du réseau. En détail, cette architecture est divisée en trois parties : les infrastructures, le contrôle et les applications. Le contrôle étant l'endroit où toutes les décisions sont prises. Le sujet de ces travaux est donc lié à l'enjeu principal de cette couche contrôle : la sécurité et la sûreté du contrôleur. En outre, les deux menaces sont considérées, mais sans les distinguer, c'est-à-dire que du point de vue de l'observateur, une défaillance du contrôleur est au même niveau qu'une compromission du contrôle. En effet, la centralisation souhaitée pour une architecture de type SDN implique que le contrôleur est un point de défaillance et d'attaque unique. En cas de défaillance du contrôleur, c'est tout le réseau qui est en panne de contrôle et de la même façon, en cas d'attaque du contrôleur l'attaquant peut mettre en place le contrôle qu'il souhaite sur tout le réseau. Dans la littérature, les architectures multicontrôleurs sont utilisées afin de répondre à ces deux problématiques. Néanmoins, ce type de contrôle distribué (physiquement ou logiquement) induit un certain nombre d'enjeux donnant lieu à l'introduction d'une interface de communication entre les contrôleurs. Cette interface est propice à la propagation des attaques puisqu'un attaquant peut répandre de mauvaises informations sur le réseau par cette interface.

C'est dans cet objectif que nous avons introduit une architecture multicontrôleurs sans interface est-ouest. Précisément, à côté du contrôleur responsable du réseau, nous ajoutons un second contrôleur dans le but d'être un observateur de ce premier contrôleur. Sa tâche est de détecter toutes anomalies dans le contrôle, qu'elles soient liées à un problème de sûreté ou à un problème de sécurité. Ce problème de détection est commun à d'autres thématiques scientifiques et nous avons dressé un état de l'art des différentes approches proposées. Une première approche consiste à établir un modèle fautif du système afin de repérer ce comportement en temps réel. Cette approche est très efficace, mais uniquement concernant la faute modélisée. Ici, on ne peut pas être exhaustif sur les menaces du contrôleur donc nous nous sommes concentrés sur la deuxième approche : construction d'un modèle non fautif et vérification que le comportement observé ne présente pas d'anomalies vis-à-vis du modèle construit. Il existe différentes façons de construire ce modèle : par connaissance experte ou bien par apprentissage. Dans cette thèse, nous avons couplé ces deux approches. Dans un premier temps, nous avons formalisé le comportement de l'activité de la commande sous la forme d'un *template* : en réponse à une requête, le contrôleur doit mettre en place un plan de données dans un certain intervalle de temps.

Cependant, cette approche n'est pas suffisante puisqu'il faut également vérifier le contenu du plan de données. Donc dans un second temps nous avons introduit des propriétés structurelles qu'un plan de données doit respecter pour être considéré comme cohérent. On a pu voir que ces propriétés sont nécessaires, mais pas suffisantes et en conséquence de ça nous avons complété la

méthode de détection par la construction d'un modèle du contrôle par apprentissage selon le type de contrôle considéré : déterministe ou non. Dans le cas déterministe, on apprend les variables internes du contrôleur à partir des décisions cohérentes qu'il prend. Il reste alors à vérifier que le contrôleur ne se contredit pas du fait de l'hypothèse de déterminisme. Cette méthode n'est toutefois plus applicable au cas non déterministe et nous avons proposé d'établir un score de vraisemblance des décisions prises par le contrôleur. Ce score est établi suivant une approche multicritères et le choix des critères dépend du cas d'application. Deux critères ont été proposés : vérification que l'impact des décisions prises par le contrôleur est vraisemblable et vérification que la séquence des décisions prises par le contrôleur est vraisemblable. Pour le premier critère, l'idée est d'évaluer la tendance de l'impact des décisions prises par le contrôleur. Pour le second critère, lors d'une phase d'apprentissage, on établit un premier modèle du comportement de l'algorithme de commande servant ensuite lors de la phase courante à évaluer le niveau de vraisemblance des décisions observées. Trois modèles ont été choisis dans cet objectif : Probabilistic Finite Automaton (PFA), Hidden Markov Model (HMM) et Recurrent Neural Network (RNN) et nous avons donc mis en place différents contrôles avec différentes variances pour voir les performances et limitations de ces trois modèles.

Nous avons pu constater que la détection depuis l'observateur, sans échanges directs avec le contrôleur, fonctionne. L'observateur réussit bien à détecter des attaques et des défaillances du contrôleur de la même façon, sans distinction et sans isolation. Nous avons confronté l'observateur à différents scénarios et différentes attaques visant le contrôleur. On a pu voir que ces attaques sont détectées dès lors que l'attaque se répercute sur l'activité de la commande. Sans cela, l'observateur ne détecte rien, mais ce qui n'est pas un problème puisque le service est toujours rendu par le contrôleur et donc le réseau n'est pas en manque de contrôle. Cependant, bien que la détection soit achevée, on notera la difficulté de prise de décision avec une certaine proportion de mauvaise décision. Cette proportion dépend des paramètres qui ont été fixés. Nous avons pu discuter de l'influence des différents paramètres et seuils en isolant leurs comportements.

Au final, cette thèse, qui s'inscrit dans le projet Digitrust Lorraine Université d'Excellence, et plus particulièrement dans la continuité des travaux antérieurs de Dorine Petit ((Petit, 2018)) et Jérémy Robert ((Robert, 2012)), matérialise et développe le concept d'observabilité de réseau. Nous sommes convaincus que ce concept est amené à se développer de par le recours de plus en plus accru à l'Intelligence Artificielle dans le contrôle des réseaux (Intelligent-Defined Networking) et par la croissance des attaques de réseau.

Par ailleurs, la méthode de détection proposée détermine si le contrôleur est attaqué, mais ne propose aucun traitement. Une proposition est que, suite à l'alarme, l'observateur doit prendre le relais du contrôleur principal comme représenté dans la Fig. 37. Plusieurs techniques peuvent être utilisées comme la phase de récupération présentée dans (Fonseca et al., 2012) et qui s'articule en deux temps : arrêter la communication entre les commutateurs et le contrôleur défaillant, puis les commutateurs cherchent à contacter l'observateur en tant que contrôleur.

Ces travaux présentent plusieurs autres perspectives, notamment afin de compléter la méthode de détection. Tout d'abord, l'application de la détection est propre au contrôle considéré et cette thèse ne traite que du cas de contrôle déterministe/non déterministe. Pour la généraliser, il faudrait prendre en compte les autres spécificités qu'un contrôle peut avoir. En effet, ces travaux posent les fondations de la méthode de détection qui devront être complétée selon les cas d'étude voir étendues pour d'autres types de contrôle. Il s'agit d'un problème de passage à l'échelle et notre méthode pourra présenter des limitations, notamment l'algorithme de détection pour un

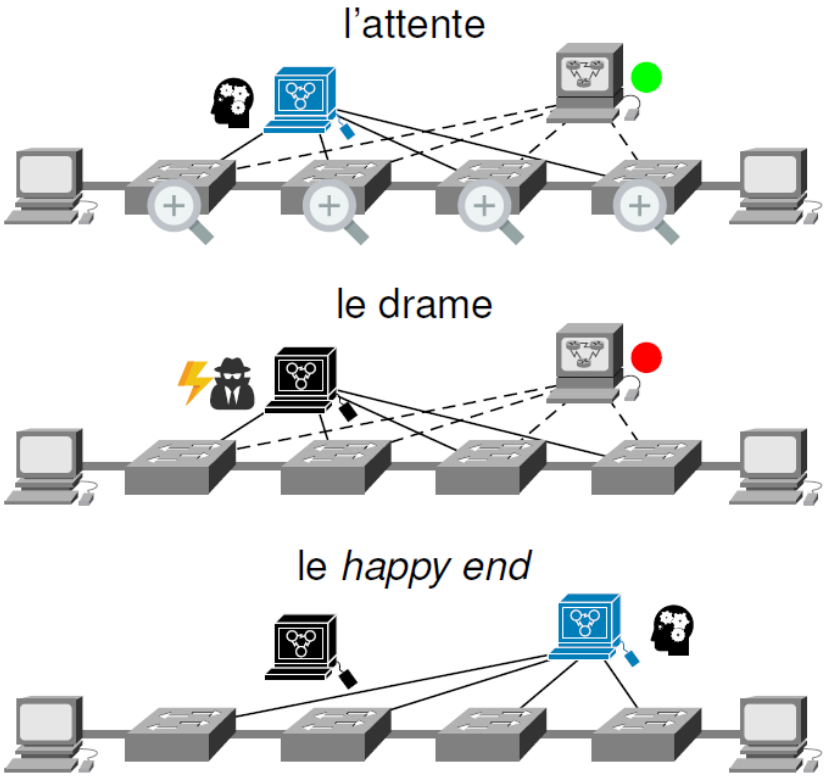


FIGURE 37 – Scénario complet après la détection pour tendre vers une automatisation sûre et sécurisée de la couche contrôle.

cas déterministe. En l'état, l'algorithme possède un bloc par comportement attendu du contrôleur. Par conséquent, en complexifiant le contrôleur, on complexifie l'observateur. Par exemple, en considérant un contrôle distribué, la méthode de détection devra prendre en considération les problématiques spécifiques liées à ce type d'architecture.

De plus, avoir un unique observateur laisse une faiblesse dans l'architecture puisqu'il est seul en charge de la détection. Il faudrait déterminer le niveau de redondance nécessaire et s'il est nécessaire de rajouter un observateur d'observateur ou non. Par ailleurs, des obstacles (notamment les problématiques d'observabilités) peuvent entraver cette détection. En effet, nous avons considéré que l'observateur n'avait aucun problème d'observabilité : c'est-à-dire qu'il avait accès à tous les paquets échangés au niveau de l'interface sud. Il se peut que quelques paquets lui échappent et auquel cas que faire ? Ce phénomène peut être dû à un problème physique ou même à un attaquant qui a détecté la présence de l'observateur. Il faudra donc trouver une solution pour de telles situations. Pour généraliser ces problèmes d'observabilité, le chiffrement se développe de plus en plus pour assurer une transmission de données sécurisée. Cela se développe au sein même de l'architecture SDN au niveau des différentes infrastructures et notamment l'interface sud (entre les infrastructures et le contrôleur). Ce chiffrement permet de sécuriser l'interface sud, mais dans le cas où l'observateur ne possède pas la clé et un tel chiffrement compliquerait la tâche de l'observateur. Il faudrait ainsi développer des mécanismes pour compléter la méthode de détection et éventuellement lui permettre de travailler à partir d'une variété réduite d'informations.

Liste des publications

Revues internationales

1. Desgeorges, L., Piriou, P.-Y., Lemattre, T., & Chraibi, H. (2021). *Formalism and semantics of pycatshoo : A simulator of distributed stochastic hybrid automata.* Reliability Engineering & System Safety, 208. Elsevier

- Desgeorges, L., Georges, J.-P., & Divoux, T. *Implementation of a SDN architecture observer : detection of failure, distributed denial-of-service, and unauthorized intrusion.* Soumis en 2022 à Security and Communication Networks. Hindawi
- Desgeorges, L., Georges, J.-P., & Divoux, T. *Detection of anomaly of a non-deterministic Software-Defined Networking control.* Soumis en 2022 à Reliability Engineering & System Safety. Elsevier

Communications dans des conférences internationales avec actes

1. Desgeorges, L., Georges, J.-P., & Divoux, T. (2022). *Detection of cyber-attacks in network control planes using Hidden Markov Model.* Accepté, à paraître dans 2022 16th IFAC Workshop on Discrete Event Systems (WODES).
2. Desgeorges, L., Georges, J.-P., & Divoux, T. (2022). *Multi-criteria detection method for non-determinist SDN control.* Dans 2022 6th IFAC Symposium on Telematics Applications (TA).
3. Desgeorges, L., Georges, J.-P., & Divoux, T. (2021). *A technique to monitor threats in SDN data plane computation.* Dans 2021 IEEE 22nd international conference on High Performance Switching and Routing (HPSR). **Prix : IEEE General Chair Best Paper Award**
4. Desgeorges, L., Georges, J.-P., & Divoux, T. (2021). *Detection of security and safety threats related to the control of a SDN architecture.* Dans 2021 14th IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in control (CESCIT).

Communications nationales sans actes

1. Desgeorges, L., Georges, J.-P., & Divoux, T. (2022). *Une méthode de détection multicritère pour un contrôle SDN non déterministe.* Journées de printemps de la SAGIP. Mai 2022
2. Desgeorges, L., Georges, J.-P., & Divoux, T. (2022). *Détection de menaces de sécurité et fiabilité visant le contrôle d'un réseau SDN.* Journées du GDR Réseaux et Systèmes Distribués. Avril 2022

3. Desgeorges, L., Georges, J.-P., & Divoux, T. (2020). *Détection de défaillances et d'attaques d'un contrôleur SDN.* Journées Nationales Automatique de la SAGIP et demi-journée GDR MACS. Novembre 2020

Livrables CRAN/CNES

1. Desgeorges, L., Hossain, M.M., Georges, J.-P., & Divoux, T. (2020). *Exploration des amendements "Time Sensitive Networking" : benchmark sur plateforme expérimentale.* R&T CRAN - CNES n°5. Novembre 2020
2. Desgeorges, L., Hossain, M.M., Georges, J.-P., & Divoux, T. (2020). *Principe et opportunités d'évaluation des amendements TSN IEEE802.1AS et IEEE802.1Qbv.* R&T CRAN - CNES n°5. Juillet 2020
3. Desgeorges, L., Hossain, M.M., Georges, J.-P., & Divoux, T. (2019). *Présentation de Time Sensitive Networking (TSN).* R&T CRAN - CNES n°5. Octobre 2019

Références

Abou El Houda, Z., Hafid, A., & Khoukhi, L. (2020). Brainchain-a machine learning approach for protecting blockchain applications using sdn. In *Icc 2020-2020 ieee international conference on communications (icc)* (pp. 1–6).

Abubakar, A., & Pranggono, B. (2017). Machine learning based intrusion detection system for software defined networks. In *2017 seventh international conference on emerging security technologies (est)* (pp. 138–143).

Ahmad, A., Harjula, E., Ylianttila, M., & Ahmad, I. (2020). Evaluation of machine learning techniques for security in sdn. In *2020 ieee globecom workshops (gc wkshps* (pp. 1–6).

Ahmad, I., Namal, S., Ylianttila, M., & Gurtov, A. (2015). Security in software defined networks : A survey. *IEEE Communications Surveys & Tutorials*, *17*(4), 2317–2346.

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system : A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, *32*(1), e4150.

Ahmed, M., Mahmood, A. N., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, *60*, 19–31.

Ahmim, A., Maglaras, L., Ferrag, M. A., Derdour, M., & Janicke, H. (2019). A novel hierarchical intrusion detection system based on decision tree and rules-based models. In *2019 15th international conference on distributed computing in sensor systems (dcoss)* (pp. 228–233).

Alharbi, T. (2020). Deployment of blockchain technology in software defined networks : A survey. *IEEE Access*, *8*, 9146–9156.

Al Hasrouty, C., Lamali, M. L., Magoni, D., & Murphy, J. (2017). Sdn-enabled adaptation of videoconference streams to network dynamics. In *Globecom 2017-2017 ieee global communications conference* (pp. 1–6).

Allen, J., Christie, A., Fithen, W., McHugh, J., & Pickel, J. (2000). *State of the practice of intrusion detection technologies* (Rapport technique). Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

Al-Rushdan, H., Shurman, M., Alnabelsi, S. H., & Althebyan, Q. (2019). Zero-day attack detection and prevention in software-defined networks. In *2019 international arab conference on information technology (acit)* (pp. 278–282).

Alto, P. (April 2012). “software-defined networking : The new norm for networks" available : https ://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/ , dernière visite le 10/01/2022 [Manuel de logiciel].

Bannour, F., Souihi, S., & Mellouk, A. (2017). Distributed sdn control : Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, *20*(1), 333–354.

Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, *37*(6), 1554–1563.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, *41*(1), 164–171.

Benson, T., Akella, A., & Maltz, D. A. (2009). Unraveling the complexity of network management. In *Nsdi* (pp. 335–348).

Benton, K., Camp, L. J., & Small, C. (2013). Openflow vulnerability assessment. In *Proceedings of the second acm sigcomm workshop on hot topics in software defined networking* (pp. 151–152).

Bertaux, L., Medjiah, S., Berthou, P., Abdellatif, S., Hakiri, A., Gelard, P., ... Bruyere, M. (2015). Software defined networking and virtualization for broadband satellite networks. *IEEE Communications Magazine*, *53*(3), 54–60.

Bertrand, N., Bouyer, P., Brihaye, T., Menet, Q., Baier, C., Größer, M., & Jurdzinski, M. (2014). Stochastic timed automata. *arXiv preprint arXiv :1410.2128*.

Bholowalia, P., & Kumar, A. (2014). Ebk-means : A clustering technique based on elbow method and k-means in wsn. *International Journal of Computer Applications*, *105*(9).

Blial, O., Ben Mamoun, M., & Benaini, R. (2016). An overview on sdn architectures with multiple controllers. *Journal of Computer Networks and Communications*, *2016*.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., & Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking : evolution, applications and research opportunities. *Journal of Internet Services and Applications*, *9*(1), 1–99.

Braun, W., & Menth, M. (2014). Software-defined networking using openflow : Protocols, applications and architectural design choices. *Future Internet*, *6*(2), 302–336.

Buckland, M., & Gey, F. (1994). The relationship between recall and precision. *Journal of the American society for information science*, *45*(1), 12–19.

Cai, Z., Cox, A. L., & Ng, T. (2010). *Maestro : A system for scalable openflow control* (Rapport technique).

Calvert, K. (2006). Reflections on network architecture : an active networking perspective. *ACM SIGCOMM Computer Communication Review*, *36*(2), 27–30.

Cambiaso, E., Chiola, G., & Aiello, M. (2019). Introducing the slowdrop attack. *Computer Networks*, *150*, 234–249.

Cappé, O., Moulines, E., & Rydén, T. (2009). Inference in hidden markov models. In *Proceedings of eusflat conference* (pp. 14–16).

Casas-Velasco, D. M. (2020). https ://github.com/danielacasasv/rsir-reinforcement-learning-and-sdn-intelligent-routing.git , dernière visite le 10/01/2022 [Manuel de logiciel].

Casas-Velasco, D. M., Rendon, O. M. C., & da Fonseca, N. L. (2020). Intelligent routing based on reinforcement learning for software-defined networking. *IEEE Transactions on Network and Service Management*, *18*(1), 870–881.

Cello, M., Xu, Y., Walid, A., Wilfong, G., Chao, H. J., & Marchese, M. (2017). Balcon : A distributed elastic sdn control via efficient switch migration. In *2017 ieee international conference on cloud engineering (ic2e)* (pp. 40–50).

CERT. (2021). Disponible : https ://www.sei.cmu.edu/about/divisions/cert/index.cfm , dernière visite 29/06/2022 [Manuel de logiciel].

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection : A survey. *ACM computing surveys (CSUR)*, *41*(3), 1–58.

Chen, C.-M., Guan, D.-J., Huang, Y.-Z., & Ou, Y.-H. (2016). Anomaly network intrusion detection using hidden markov model. *Int. J. Innov. Comput. Inform. Control*, *12*, 569–580.

Chica, J. C. C., Imbachi, J. C., & Vega, J. F. B. (2020). Security in sdn : A comprehensive survey. *Journal of Network and Computer Applications*, *159*, 102595.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Das, R. K., Pohrmen, F. H., Maji, A. K., & Saha, G. (2020). Ft-sdn : a fault-tolerant distributed architecture for software defined network. *Wireless personal communications*, *114*(2), 1045–1066.

Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on machine learning* (pp. 233–240).

de Miranda Rios, V., Inácio, P. R., Magoni, D., & Freire, M. M. (2021). Detection of reduction-of-quality ddos attacks using fuzzy logic and machine learning algorithms. *Computer Networks*, *186*, 107792.

Deng, S., Gao, X., Lu, Z., Li, Z., & Gao, X. (2019). Dos vulnerabilities and mitigation strategies in software-defined networks. *Journal of Network and Computer Applications*, *125*, 209-219. Consulté sur `https://www.sciencedirect.com/science/article/pii/S1084804518303333` doi: https://doi.org/10.1016/j.jnca.2018.10.011

De Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., & Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. In *2014 ieee colombian conference on communications and computing (colcom)* (pp. 1–6).

Derhab, A., Guerroumi, M., Belaoued, M., & Cheikhrouhou, O. (2021). Bmc-sdn : blockchain-based multicontroller architecture for secure software-defined networks. *Wireless Communications and Mobile Computing*, *2021*.

Devarakonda, N., Pamidi, S., Kumari, V. V., & Govardhan, A. (2012). Intrusion detection system using bayesian network and hidden markov model. *Procedia Technology*, *4*, 506–514.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, *29*(2), 103–130.

Feamster, N., Rexford, J., & Zegura, E. (2013, 12). The road to sdn. *Queue*, *11*. doi: 10.1145/2559899.2560327

Fonseca, P., Bennesby, R., Mota, E., & Passito, A. (2012). A replication component for resilient openflow-based networking. In *2012 ieee network operations and management symposium* (pp. 933–939).

Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268–278.

Fouladi, R. F., Kayatas, C. E., & Anarim, E. (2016). Frequency based ddos attack detection approach using naive bayes classification. In *2016 39th international conference on telecommunications and signal processing (tsp)* (pp. 104–107).

Foundation, O. N. (2011). `https://opennetworking.org/` , dernière visite le 10/01/2022 [Manuel de logiciel].

Fouquet, K., Faraut, G., & Lesage, J.-J. (2020). Life habits modeling with stochastic timed automata in ambient assisted living. In *2020 ieee international conference on systems, man, and cybernetics (smc)* (pp. 2740–2745).

Georges, J.-P. (2019). *Performance evaluation & network automation. For a triptych QoS, QoE, QiS* (Habilitation à diriger des recherches, Université de Lorraine). Consulté sur `https://hal.archives-ouvertes.fr/tel-02419132`

Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., & Wilcox, J. (2011). Intelligent design enables architectural evolution. In *Proceedings of the 10th acm workshop on hot topics in networks* (pp. 1–6).

Gkountis, C., Taha, M., Lloret, J., & Kambourakis, G. (2017). Lightweight algorithm for protecting sdn controller against ddos attacks. In *2017 10th ifip wireless and mobile networking conference (wmnc)* (pp. 1–6).

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., & Shenker, S. (2008). Nox : towards an operating system for networks. *ACM SIGCOMM computer communication review*, *38*(3), 105–110.

Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S., & Chao, H. J. (2014). Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks*, *68*, 95–109.

Hafeez, T., Ahmed, N., Ahmed, B., & Malik, A. W. (2017). Detection and mitigation of congestion in sdn enabled data center networks : A survey. *IEEE Access*, *6*, 1730–1740.

Hakiri, A., Berthou, P., Gokhale, A., & Abdellatif, S. (2015). Publish/subscribe-enabled software defined networking for efficient and scalable iot communications. *IEEE communications magazine*, *53*(9), 48–54.

Haleplidis, E., Hadi Salim, J., Halpern, J. M., Hares, S., Pentikousis, K., Ogawa, K., ... Koufopavlou, O. (2015). Network programmability with forces. *IEEE Communications Surveys & Tutorials*, *17*(3), 1423-1440. doi: 10.1109/COMST.2015.2439033

Halpern, J. M., HAAS, R., avri doria, Dong, L., Wang, W., Khosravi, H. M., ... Gopal, R. (2010, mars). *Forwarding and control element separation (forces) protocol specification* (N° 5810). RFC 5810. RFC Editor. Consulté sur `https://www.rfc-editor.org/info/rfc5810` doi: 10.17487/RFC5810

Hassas Yeganeh, S., & Ganjali, Y. (2012). Kandoo : a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on hot topics in software defined networks* (pp. 19–24).

Heller, B., Sherwood, R., & McKeown, N. (2012). The controller placement problem. *ACM SIGCOMM Computer Communication Review*, *42*(4), 473–478.

Herrera, J. A., & Camargo, J. E. (2019). A survey on machine learning applications for software defined network security. In *International conference on applied cryptography and network security* (pp. 70–93).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hogg, S. (2014). Sdn security attack vectors and sdn hardening. *Network World*.

Holgado, P., Villagrá, V. A., & Vazquez, L. (2017). Real-time multistep attack prediction based on hidden markov models. *IEEE Transactions on Dependable and Secure Computing*, *17*(1), 134–147.

Hong, S., Xu, L., Wang, H., & Gu, G. (2015). Poisoning network visibility in software-defined networks : New attacks and countermeasures..

Hosny, K. M., Gouda, A. E.-S., & Mohamed, E. R. (2020). New detection mechanism for distributed denial of service attacks in software defined networks. *International Journal of Sociotechnology and Knowledge Development (IJSKD)*, *12*(2), 1–30.

Hu, T., Guo, Z., Yi, P., Baker, T., & Lan, J. (2018). Multi-controller based software-defined networking : A survey. *IEEE access*, *6*, 15980–15996.

Hubballi, N., & Suryanarayanan, V. (2014). False alarm minimization techniques in signature-based intrusion detection systems : A survey. *Computer Communications*, *49*, 1–17.

Hurley, T., Perdomo, J. E., & Perez-Pons, A. (2016). Hmm-based intrusion detection system for software defined networking. In *2016 15th ieee international conference on machine learning and applications (icmla)* (pp. 617–621).

Hyder, M. F., & Ismail, M. A. (2021). Securing control and data planes from reconnaissance attacks using distributed shadow controllers, reactive and proactive approaches. *IEEE Access*, *9*, 21881–21894.

Ingre, B., Yadav, A., & Soni, A. K. (2017). Decision tree based intrusion detection system for nsl-kdd dataset. In *International conference on information and communication technology for intelligent systems* (pp. 207–218).

Jabbar, M., Aluvalu, R., & Reddy, S. S. S. (2017). Intrusion detection system using bayesian network and feature subset selection. In *2017 ieee international conference on computational intelligence and computing research (iccic)* (pp. 1–5).

Jacobson, V. (1988). Congestion avoidance and control. *ACM SIGCOMM computer communication review*, *18*(4), 314–329.

Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking : State of the art and research challenges. *Computer Networks*, *72*, 74–98.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41–50.

Kim, J., Kim, J., Thu, H. L. T., & Kim, H. (2016). Long short term memory recurrent neural network classifier for intrusion detection. In *2016 international conference on platform technology and service (platcon)* (pp. 1–5).

Kirkpatrick, K. (2013). Software-defined networking. *Communications of the ACM*, *56*(9), 16–19.

Ko, C., Ruschitzka, M., & Levitt, K. (1997). Execution monitoring of security-critical programs in distributed systems : A specification-based approach. In *Proceedings. 1997 ieee symposium on security and privacy (cat. no. 97cb36097)* (pp. 175–187).

Kolbæk, M., Yu, D., Tan, Z.-H., & Jensen, J. (2017). Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *25*(10), 1901–1913.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., ... others (2010). Onix : A distributed control platform for large-scale production networks. In *9th usenix symposium on operating systems design and implementation (osdi 10)*.

Kreutz, D., Ramos, F. M., & Verissimo, P. (2013). Towards secure and dependable software-defined networks. In *Proceedings of the second acm sigcomm workshop on hot topics in software defined networking* (pp. 55–60).

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking : A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76.

Ksentini, A., Bagaa, M., Taleb, T., & Balasingham, I. (2016). On using bargaining game for optimal placement of sdn controllers. In *2016 ieee international conference on communications (icc)* (pp. 1–6).

Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I., & Kim, K. J. (2019). A survey of deep learning-based network anomaly detection. *Cluster Computing*, *22*(1), 949–961.

Lam, J.-H., Lee, S.-G., Lee, H.-J., & Oktian, Y. E. (2015). Securing distributed sdn with ibc. In *2015 seventh international conference on ubiquitous and future networks* (pp. 921–925).

Laterrasse, J., Chatzis, K., & Coutard, O. (1992). La problématique centralisation/décentralisation : architecture des réseaux et choix organisationnels. *FLUX Cahiers scientifiques internationaux Réseaux et Territoires*, *8*(8), 47–53.

Lee, S., Yoon, C., Lee, C., Shin, S., Yegneswaran, V., & Porras, P. A. (2017a). Delta : A security assessment framework for software-defined networks. In *Ndss.*

Lee, S., Yoon, C., Lee, C., Shin, S., Yegneswaran, V., & Porras, P. A. (2017b). Delta : A security assessment framework for software-defined networks. In *Ndss.*

Lee, S., Yoon, C., & Shin, S. (2016). The smaller, the shrewder : A simple malicious application can kill an entire sdn environment. In *Proceedings of the 2016 acm international workshop on security in software defined networks & network function virtualization* (pp. 23–28).

Levin, D., Wundsam, A., Heller, B., Handigol, N., & Feldmann, A. (2012). Logically centralized ? state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on hot topics in software defined networks* (pp. 1–6).

Li, Y., Tong, Y., & Giua, A. (2020). Detection and prevention of cyber-attacks in networked control systems. *IFAC-PapersOnLine*, *53*(4), 7–13.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013a). Intrusion detection system : A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16–24.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013b). Intrusion detection system : A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16–24.

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern recognition*, *36*(2), 451–461.

Lindemann, B., Maschler, B., Sahlab, N., & Weyrich, M. (2021). A survey on anomaly detection for technical systems using lstm networks. *Computers in Industry*, *131*, 103498.

Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems : A survey. *applied sciences*, *9*(20), 4396.

Liu, J., Li, Q., Chen, W., Yan, Y., Qiu, Y., & Cao, T. (2019). Remaining useful life prediction of pemfc based on long short-term memory recurrent neural networks. *International Journal of Hydrogen Energy*, *44*(11), 5470–5480.

Liu, X., Xue, H., Feng, X., & Dai, Y. (2011). Design of the multi-level security network switch system which restricts covert channel. In *2011 ieee 3rd international conference on communication software and networks* (pp. 233–237).

Liyanage, M., Ahmed, I., Okwuibe, J., Ylianttila, M., Kabir, H., Santos, J. L., ... De Oca, E. M. (2017). Enhancing security of software defined mobile networks. *IEEE Access*, *5*, 9422–9438.

Macedo, R., de Castro, R., Santos, A., Ghamri-Doudane, Y., & Nogueira, M. (2016). Self-organized sdn controller cluster conformations against ddos attacks effects. In *2016 ieee global communications conference (globecom)* (pp. 1–6).

Madhulatha, T. S. (2012). An overview on clustering methods. *arXiv preprint arXiv :1205.1117*.

Mamushiane, L., Lysko, A., & Dlamini, S. (2018). A comparative evaluation of the performance of popular sdn controllers. In *2018 wireless days (wd)* (pp. 54–59).

Mandal, S. (2015). Experience with b4 : Google's private {SDN} backbone.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... Turner, J. (2008). Openflow : enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, *38*(2), 69–74.

Melazzi, N. B., Detti, A., Mazza, G., Morabito, G., Salsano, S., & Veltri, L. (2012). An openflow-based testbed for information centric networking. In *2012 future network & mobile summit (futurenetw)* (pp. 1–9).

Mostafavi, S., Hakami, V., & Paydar, F. (2020). Performance evaluation of software-defined networking controllers : A comparative study. *Computer and Knowledge Engineering*, *2*(2), 63–73.

Muruti, G., Rahim, F. A., & bin Ibrahim, Z.-A. (2018). A survey on anomalies detection techniques and measurement methods. In *2018 ieee conference on application, information and network security (ains)* (pp. 81–86).

Nam Nguyen, H., Anh Tran, H., Fowler, S., & Souihi, S. (2021). A survey of blockchain technologies applied to software-defined networking : Research challenges and solutions. *IET Wireless Sensor Systems*, *11*(6), 233–247.

Nanda, S., Zafari, F., DeCusatis, C., Wedaa, E., & Yang, B. (2016). Predicting network attack patterns in sdn using machine learning approach. In *2016 ieee conference on network function virtualization and software defined networks (nfv-sdn)* (pp. 167–172).

Nassif, A. B., Talib, M. A., Nasir, Q., & Dakalbab, F. M. (2021). Machine learning for anomaly detection : A systematic review. *Ieee Access*, *9*, 78658–78700.

Nayyar, S., Arora, S., & Singh, M. (2020). Recurrent neural network based intrusion detection system. In *2020 international conference on communication and signal processing (iccsp)* (pp. 0136–0140).

Niyaz, Q., Sun, W., & Javaid, A. Y. (2016). A deep learning based ddos detection system in software-defined networking (sdn). *arXiv preprint arXiv :1611.07400*.

Nweke, L. O. (2021). A survey of specification-based intrusion detection techniques for cyber-physical systems. *International Journal of Advanced Computer Science and Applications*, *12*(5).

ONF. (2017). https ://opennetworking.org/onos/, dernière visite le 07/09/2021 [Manuel de logiciel].

ONF. (Juin, 2012). Openflow specification v1.3 https ://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf, dernière visite le 14/09/2021 [Manuel de logiciel].

Pandalai, D. N., & Holloway, L. E. (2000). Template languages for fault monitoring of timed discrete event processes. *IEEE transactions on automatic control*, *45*(5), 868–882.

Pascoal, T. A., Fonseca, I. E., & Nigam, V. (2020). Slow denial-of-service attacks on software defined networks. *Computer Networks*, *173*, 107223.

Pashkov, V., Shalimov, A., & Smeliansky, R. (2014). Controller failover for sdn enterprise networks. In *2014 international science and technology conference (modern networking technologies)(monetec)* (pp. 1–6).

Paxson, V. (1999). Bro : a system for detecting network intruders in real-time. *Computer networks*, *31*(23-24), 2435–2463.

Petit, D. (2018). *Unification des stratégies de contrôle de réseau embarqué temps-réel reconfigurable* (Thèse de doctorat non publiée). Université de Lorraine.

Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., & Gu, G. (2012). A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on hot topics in software defined networks* (p. 121–126). New York, NY, USA : Association for Computing Machinery. Consulté sur `https://doi.org/10.1145/2342441.2342466` doi: 10.1145/2342441.2342466

Porras, P. A., Cheung, S., Fong, M. W., Skinner, K., & Yegneswaran, V. (2015). Securing the software defined network control layer. In *Ndss*.

Qi, C., Wu, J., Hu, H., Cheng, G., Liu, W., Ai, J., & Yang, C. (2016). An intensive security architecture with multi-controller for sdn. In *2016 ieee conference on computer communications workshops (infocom wkshps)* (pp. 401–402).

Qin, P., Dai, B., Huang, B., & Xu, G. (2015). Bandwidth-aware scheduling with sdn in hadoop : A new trend for big data. *IEEE Systems Journal*, *11*(4), 2337–2344.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81–106.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Ramos, F. M., Kreutz, D., & Verissimo, P. (2015). Software-defined networks : On the road to the softwarization of networking. *Cutter IT journal*.

Rathore, S., Wook Kwon, B., & Park, J. H. (2019). Blockseciotnet : Blockchain-based decentralized security architecture for iot network. *Journal of Network and Computer Applications*, *143*, 167-177. Consulté sur `https://www.sciencedirect.com/science/article/pii/S1084804519302243` doi: https://doi.org/10.1016/j.jnca.2019.06.019

Robert, J. (2012). *De l'usage d'architectures ethernet commutées embarquées dans les lanceurs spatiaux* (Thèse de doctorat non publiée). Université de Lorraine.

Salaün, A., Petetin, Y., & Desbouvries, F. (2019). Comparing the modeling powers of rnn and hmm. In *2019 18th ieee international conference on machine learning and applications (icmla)* (pp. 1496–1499).

Sallahi, A., & St-Hilaire, M. (2014). Optimal model for the controller placement problem in software defined networks. *IEEE communications letters*, *19*(1), 30–33.

Samaan, N., & Karmouch, A. (2009). Towards autonomic network management : an analysis of current and future research directions. *IEEE Communications Surveys & Tutorials*, *11*(3), 22–36.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. C. (1996). Failure diagnosis using discrete-event models. *IEEE transactions on control systems technology*, *4*(2), 105–124.

Sanyal, S., Barai, M. K., & Goplani, A. (2021). A novel blockchain based software defined network (sdn) architecture to curb the impact of dos/ddos.

Sarker, I. H. (2021). Deep learning : a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, *2*(6), 1–20.

Scarfone, K., Mell, P., et al. (2007). Guide to intrusion detection and prevention systems (idps). *NIST special publication*, *800*(2007), 94.

Scott-Hayward, S., O'Callaghan, G., & Sezer, S. (2013). Sdn security : A survey. In *2013 ieee sdn for future networks and services (sdn4fns)* (pp. 1–7).

Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., & Zhou, S. (2002). Specification-based anomaly detection : a new approach for detecting network intrusions. In *Proceedings of the 9th acm conference on computer and communications security* (pp. 265–274).

Sellami, B., Hakiri, A., Yahia, S. B., & Berthou, P. (2022). Energy-aware task scheduling and offloading using deep reinforcement learning in sdn-enabled iot network. *Computer Networks*, *210*, 108957.

Selvi, H., Gür, G., & Alagöz, F. (2016). Cooperative load balancing for hierarchical sdn controllers. In *2016 ieee 17th international conference on high performance switching and routing (hpsr)* (pp. 100–105).

Shang, F., Li, Y., Fu, Q., Wang, W., Feng, J., & He, L. (2018). Distributed controllers multi-granularity security communication mechanism for software-defined networking. *Computers & Electrical Engineering*, *66*, 388–406.

Shin, S., Xu, L., Hong, S., & Gu, G. (2016). Enhancing network security through software defined networking (sdn). In *2016 25th international conference on computer communication and networks (icccn)* (pp. 1–9).

Shu, Z., Wan, J., Li, D., Lin, J., Vasilakos, A. V., & Imran, M. (2016). Security in software-defined networking : Threats and countermeasures. *Mobile Networks and Applications*, *21*(5), 764–776.

Šimundić, A.-M. (2009). Measures of diagnostic accuracy : basic definitions. *ejifcc*, *19*(4), 203.

Singh, J., & Behal, S. (2020). Detection and mitigation of ddos attacks in sdn : A comprehensive review, research challenges and future directions. *Computer Science Review*, *37*, 100279.

Swami, R., Dave, M., & Ranga, V. (2019). Software-defined networking-based ddos defense mechanisms. *ACM Computing Surveys (CSUR)*, *52*(2), 1–36.

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2018). Deep recurrent neural network for intrusion detection in sdn-based networks. In *2018 4th ieee conference on network softwarization and workshops (netsoft)* (pp. 202–206).

Team, R. P. (2012). Available : https ://www.opennetworking.org/ , dernière visite le 10/01/2022 [Manuel de logiciel].

Teng, S., Wu, N., Zhu, H., Teng, L., & Zhang, W. (2017). Svm-dt-based adaptive and collaborative intrusion detection. *IEEE/CAA Journal of Automatica Sinica*, *5*(1), 108–118.

Tootoonchian, A., & Ganjali, Y. (2010). Hyperflow : A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on research on enterprise networking* (Vol. 3).

TOTEM. (January, 2006). https ://totem.info.ucl.ac.be/index.html , dernière visite le 10/01/2022 [Manuel de logiciel].

Tripathi, N., & Hubballi, N. (2018). Slow rate denial of service attacks against http/2 and detection. *Computers & security*, *72*, 255–272.

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning : A review. *expert systems with applications*, *36*(10), 11994–12000.

Tseng, C.-Y., Balasubramanyam, P., Ko, C., Limprasittiporn, R., Rowe, J., & Levitt, K. (2003). A specification-based intrusion detection system for aodv. In *Proceedings of the 1st acm workshop on security of ad hoc and sensor networks* (pp. 125–134).

Uppuluri, P., & Sekar, R. (2001). Experiences with specification-based intrusion detection. In *International workshop on recent advances in intrusion detection* (pp. 172–189).

Vaccari, I., Aiello, M., & Cambiaso, E. (2020). Slowtt : A slow denial of service against iot networks. *Information*, *11*(9), 452.

Valadarsky, A., Schapira, M., Shahaf, D., & Tamar, A. (2017). Learning to route. In *Proceedings of the 16th acm workshop on hot topics in networks* (pp. 185–191).

Viterbi, A. (1971). Convolutional codes and their performance in communication systems. *IEEE Transactions on Communication Technology*, *19*(5), 751–772.

Vizarreta, P., Heegaard, P., Helvik, B., Kellerer, W., & Machuca, C. M. (2017). Characterization of failure dynamics in sdn controllers. In *2017 9th international workshop on resilient networks design and modeling (rndm)* (pp. 1–7).

Wang, W., Ke, X., & Wang, L. (2018). A hmm-r approach to detect l-ddos attack adaptively on sdn controller. *Future Internet*, *10*(9), 83.

Wang, Y., Hu, T., Tang, G., Xie, J., & Lu, J. (2019). Sgs : Safe-guard scheme for protecting control plane against ddos attacks in software-defined networking. *IEEE Access*, *7*, 34699–34710.

Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2014). A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, *17*(1), 27–51.

Xie, J., Yu, F. R., Huang, T., Xie, R., Liu, J., Wang, C., & Liu, Y. (2018). A survey of machine learning techniques applied to software defined networking (sdn) : Research issues and challenges. *IEEE Communications Surveys & Tutorials*, *21*(1), 393–430.

Yang, H., Liang, Y., Yuan, J., Yao, Q., Yu, A., & Zhang, J. (2020). Distributed blockchain-based trusted multidomain collaboration for mobile edge computing in 5g and beyond. *IEEE Transactions on Industrial Informatics*, *16*(11), 7094–7104.

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, *5*, 21954–21961.

Yonghong, F., Jun, B., Jianping, W., Ze, C., Ke, W., & Min, L. (2014). A dormant multi-controller model for software defined networking. *China Communications*, *11*(3), 45–55.

Yoon, C., Lee, S., Kang, H., Park, T., Shin, S., Yegneswaran, V., ... Gu, G. (2017). Flow wars : Systemizing the attack surface and defenses in software-defined networks. *IEEE/ACM Transactions on Networking*, *25*(6), 3514–3530.

Zakia, U., & Yedder, H. B. (2017). Dynamic load balancing in sdn-based data center networks. In *2017 8th ieee annual information technology, electronics and mobile communication conference (iemcon)* (pp. 242–247).

Zhang, G. P. (2000). Neural networks for classification : a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *30*(4), 451–462.

Zheng, J., Li, Q., Gu, G., Cao, J., Yau, D. K., & Wu, J. (2018). Realtime ddos defense using cots sdn switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security*, *13*(7), 1838–1853.

Zhou, W., Jin, D., Croft, J., Caesar, M., & Godfrey, P. B. (2015). Enforcing customizable consistency properties in {Software-Defined} networks. In *12th usenix symposium on networked systems design and implementation (nsdi 15)* (pp. 73–85).

Résumé

L'architecture Software-Defined Networking (SDN) a été introduite dans l'objectif de proposer un contrôle centralisé. En conséquence, une seule entité est en charge du contrôle ce qui rend indispensable de garantir un contrôle sûr et sécurisé. On peut trouver dans la littérature des solutions multicontrôleurs renforçant la couche contrôle contre ces menaces. Cependant une telle architecture amène de nouvelles spécificités et pour assurer la cohérence entre les contrôleurs, une interface de communication entre eux est nécessaire. Cette interface constitue une menace pour la sécurité puisqu'un attaquant peut propager des informations malveillantes et erronées sur le réseau aux autres contrôleurs. Dans cet objectif, ces travaux visent à introduire une architecture de contrôle. Cette architecture est composée d'un contrôleur nominal et un observateur en charge de la détection d'anomalies dans les décisions prises par le contrôleur. Pour cela, seule l'activité de la commande est étudiée. Des spécifications, des propriétés temporelles et structurelles et enfin des modèles d'analyse de la vraisemblance des solutions sont proposés et comparés. Pour cela, un score de vraisemblance est associé aux observations et est déterminé selon une approche multicritères. Ici, deux critères ont été proposés : la performance des plans et la vraisemblance des routes. Les performances obtenues montrent que les méthodes proposées sont applicables, mais présentent des limites. De plus, ces travaux posent les fondements d'une méthode de détection et plus généralement ceux d'un concept d'observabilité de réseau.

Mots-clés: Contrôle de réseau, Sécurité, Sûreté, Détection d'Anomalies, Observabilité.

Abstract

The Software-Defined Networking (SDN) architecture was introduced with the objective of providing centralized control. As a result, only one entity is in charge of control, which makes it essential to ensure safe and secure control. We can find in the literature multi-controller solutions to reinforce the control layer against these threats. However, such an architecture brings new specificities and to ensure consistency between the controllers, a communication interface between them is necessary. This interface is a security threat since an attacker can propagate malicious and erroneous information on the network to other controllers. With this objective, this work aims at introducing a control architecture. This architecture is composed of a nominal controller and an observer in charge of detecting anomalies in the decisions taken by the controller. For this, only the activity of the controller is studied. Specifications, temporal and structural properties and finally models for the analysis of the likelihood of solutions are proposed and compared. A likelihood score is associated to the observations and is determined according to a multi-criteria approach. Here, two criteria have been proposed : the performance of the plans and the likelihood of the routes. The obtained performances show that the proposed methods are applicable, but have some limitations. Moreover, this work lays the foundations of a detection method and more generally of a network observability concept.

Keywords: Network Control, Security, Safety, Detection of Anomalies, Observability.