Evolutivité d'une architecture en temps réel de filtrage d'alertes générées par les systèmes de détection d'intrusions sur les réseaux

Ahmad Faour, Philippe Leray, Bassam Eter

Evolutivité d'une architecture en temps réel de filtrage d'alertes générées par les systèmes de détection d'intrusions sur les réseaux

Evolution detection in real-time architecture of alert filtering in network intrusion detection systems

A. Faour[1,2] P. Leray[3] B. Eter[2]

[1] Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes (LITIS), EA 4108, INSA Rouen, France
[2] Laboratoire de Physique des Matériaux (LPM), Université Libanaise, Beyrouth, Liban
[3] Equipe COnnaissances et Décision (COD), Laboratoire d'informatique de Nantes Atlantique (LINA), FRE 2729, Ecole Polytechnique de l'Université de Nantes, France

ahmad.faour@ul.edu.lb philippe.leray@univ-nantes.fr beter@ul.edu.lb

Résumé

Une des difficultés majeures que rencontrent les administrateurs de sécurité en utilisant les systèmes de détection d'intrusions (NIDS) est le nombre énorme d'alertes déclenchées chaque jour. Ces limitations sont provoquées par l'absence d'un mécanisme qui peut prétraiter et filtrer le nombre massif d'alertes générées par les NIDS. Dans nos travaux passés, nous avons proposé une architecture pour filtrer les alertes générées par les NIDS. Cette architecture est une combinaison des méthodes de classification non-supervisée comme les cartes auto-organisatrices de Kohonen (SOM) et de modèles graphiques probabilistes comme les réseaux bayésiens utilisés ici pour de la classification supervisée. Cependant, l'exploitation de cette architecture en temps réel va poser plusieurs défis. Il faut tout d'abord prendre en compte l'évolution de la plate-forme surveillée (intégration de nouvelles machines, équipements réseau, ...). Il faut aussi, en second lieu, pouvoir réagir à l'apparition de nouvelles attaques mais aussi à l'évolution des comportements-types des utilisateurs. Pour résoudre ces problèmes, et plus particulièrement le dernier, nous utilisons le concept de rejet en distance et quelques tests d'hypothèses statistiques. Puis, nous proposons quatre indicateurs statistiques comme entrées d'une fonction de décision de ré-apprentissage de notre architecture. Pour finir, la validité de tous ces indicateurs est testée par des expériences sur des journaux réels d'alertes extraits d'un NIDS surveillant le réseau du Rectorat de Rouen.

Mots Clef

Ré-apprentissage, Filtrage d'alertes, Détection d'intrusions, Clustering, Rejet en distance, Tests d'hypothèses statistiques.

Abstract

It is a well-known problem that intrusion detection systems (NIDS) overload their human operators by triggering thousands of alarms per day. These limitations are caused by the absence of a mechanism that can preprocess and filter the massive number of alerts from IDS. In our past work, we proposed an architecture for filtering the alarms generated by the NIDS. This architecture is a combination of unsupervised classification methods like self-organizing maps (SOM) and probabilistic graphical models like Bayesian Networks used here in supervised classification framework. However, exploiting this architecture in real time will pose several challenges on its behavior. In this work, we underline three problems to be solved : first, the evolution of the monitored platform (integration of new machines or network equipments), second, the apparition of new attacks and third, the evolution of user behavior-types. For the resolution of these problems and especially the last one, we used the distance rejection concept and some statistical hypothesis tests. Then, we propose four statistical indicators as entries of a decision function for the re-learning of the entire system. Finally, the validity of all these indicators is tested by experiments made on real logs extracted from a NIDS that control the network of "Rectorat de Rouen".

Keywords

Relearning, Network intrusion detection, Alarm filtering, Clustering, Outlier detection, Statistical hypothesis tests.

1 Introduction

La détection des tentatives d'attaques sur un réseau est une problématique très importante dans le domaine de la sécurité informatique [1]. Les technologies classiques de protection des réseaux comme les pare-feux sont en effet inefficaces contre la plupart des attaques actuelles. Aussi sont apparus de nouveaux équipements réseaux pour prendre en compte ces carences, les NIDS, systèmes de détection d'intrusions réseaux, dont le but est de détecter les tentatives d'attaque qu'un pare-feu ne peut pas bloquer [4, 11]. Malheureusement, en pratique,les NIDS génèrent tellement d'alertes sur un réseau important qu'il en devient très difficile de déterminer celles générés par une attaque réelle. Sur un réseau de taille moyenne (entre 250 et 500 machines), plusieurs milliers d'alertes sont générées quotidiennement, rendant quasiment impossible l'exploitation des résultats. La conséquence est que l'administrateur est obligé de revoir sérieusement à la hausse son seuil de tolérance, ce qui va le conduire à passer à côté de beaucoup de problèmes réels et permettre à un pirate de haut niveau de réussir une attaque suffisamment discrète pour ne pas être détectée. Ainsi le problème principal du NIDS n'est pas de laisser passer certaines attaques (dans la pratique il en détecte la quasi totalité) mais de noyer l'administrateur sous un flot d'informations. Ce problème est un inconvénient majeur pour que le déploiement des outils de détection d'intrusion puisse être pratique. En effet, le NIDS n'est pas capable de juger de la pertinence, de la gravité et de la corrélation des attaques. Il génère tellement d'alertes qu'il va être très difficile de détecter les problèmes graves au milieu de toutes les alarmes.

Intuitivement, la manière la plus attrayante pour traiter les faux positifs est probablement de créer de "meilleurs" IDS qui soient moins enclin aux faux positifs, comme par exemple, la technologie de détecteurs de [26], les outils de détection d'attaques de serveur Web proposés par [2] ou le système de détection d'intrusions basé-réseau qui se focalise sur les attaques réseaux de bas niveau de [22].

Récemment, plusieurs techniques de corrélation d'alertes et d'analyse des scénarios d'attaques ont été proposées. Les systèmes de corrélation d'alertes (ACS) [8, 9, 10, 13, 23, 24] post-traitent les alertes des IDS en temps réel et automatisent une partie du processus de traitement de ces alertes. Les alertes peuvent représenter les étapes multiples d'un scénario d'attaque. L'essentiel réside dans le compromis effectué entre la quantité d'alertes remontées et la finesse de ces dernières. Les informations qu'elles contiennent manquent de précision et sont, de plus, parcellaires et de très bas niveau. Ces alertes sont par conséquent d'un intérêt limité pour un opérateur humain. Plus précisement, les ACS tentent de grouper les alertes de sorte que les alertes du même groupe concernent le même phénomène (i.e., la même attaque). Puis, seuls les groupes d'alerte sont expédiés à l'opérateur de sécurité [12]. De cette façon, les ACS offrent une vue plus condensée sur le problème de sécurité soulevé par les IDS. En outre, ils aident à distinguer les vraies menaces de sécurité des faux positifs.

Plus récemment, des outils comme les réseaux bayésiens, les arbres de décision ou l'analyse en composante principale ont été utilisés dans le cadre de la détection d'intrusions ou du filtrage d'alarmes [3, 5, 6].

Assez proche de ces principes, nous avons proposé dans [15] une approche de filtrage d'alertes générées par les NIDS. Cette approche est une combinaison des méthodes de classification non supervisées comme les cartes auto-organisatrices de Kohonen et supervisées comme les réseaux bayésiens. Trois points distinguent notre approche des systèmes de corrélation d'alertes. Tout d'abord, notre approche étudie aussi la détection de nouvelles stratégies d'attaque. En second lieu notre système n'utilise aucune information préalable sur les scénarios d'attaques. Troisièmement, nous visons à construire des profils de nos machines internes en utilisant les alertes générées par les NIDS en tant qu'attributs de chaque machine dans un intervalle de temps. Ces profils peuvent être employés par l'opérateur comme indicateur du comportement normal ou anormal. De cette façon, notre approche est analogue aux techniques de détection d'anomalie.

Nous reviendrons tout d'abord sur une description de notre architecture dans la section 2 en soulignant les problèmes rencontrés en cas d'exploitation en temps réel et en présentant les solutions proposées. La section 3 traite du problème d'évolution du réseau ou du NIDS. La section 4 se concentre sur le problème d'évolution des comportements types. Ensuite, la section 5 illustre le choix d'une fonction de décision de ré-apprentissage du système et les paramètres utilisés. Pour finir, la section 6 présente les expérimentations et résultats obtenus.

2 Une architecture temps réél pour le filtrage d'alarmes

2.1 Notre architecture de filtrage d'alarmes

Nous avons proposé dans [15] une architecture automatique pour le filtrage des alertes générées par les NIDS. Le but de notre système est de partir des alarmes générées par un NIDS, et d'essayer de filtrer les alarmes pour déterminer s'il y a eu une attaque sur le réseau pendant un laps de temps fixé. Notre système se décompose en trois étapes.

En premier lieu, en considérant qu'un scénario d'attaque consiste en une série d'événements se déroulant dans un intervalle de temps, nous commençons par faire une synthèse des alarmes générées par le NIDS dans une fenêtre temporelle fixée. Cette synthèse nous donne un résumé du comportement de toutes les machines externes (attaquantes ?) à destination de toutes les IP internes (attaquées).

Nous partons ensuite du principe que ce comportement peut être similaire pour plusieurs machines externes (qui tenteraient le même genre d'attaque vers une même machine interne), ou à destination de plusieurs machines internes (une même attaque pourrait être dirigée vers plusieurs machines). Nous allons donc regrouper ces com-

portements en un certain nombre de comportements-types, en utilisant une technique de classification non supervisée classique, les cartes auto-organisatrices de Kohonen (SOM) [17].
Nous pouvons maintenant faire une synthèse du nombre de comportements de chaque type ayant eu lieu à destination de chaque machine interne. Cette synthèse nous résume les différents types d'attaques potentielles visant chaque machine du réseau pendant notre fenêtre de temps. Ces informations sont alors utilisées pour déterminer si le réseau a réellement été attaqué. Cette tâche de classification a été respectivement réalisée par deux approches différentes : les réseaux bayésiens [18] et les machines à vecteurs supports.

2.2 Exploitation en temps réél

La détection d'intrusions dans les réseaux est un processus qui évolue avec le temps. Un NIDS doit pouvoir être modulable et configurable de manière à s'adapter parfaitement aux changements qui peuvent arriver au cours du temps sur les plates-formes et architectures réseaux qu'il surveille et sur lesquelles il doit parallèlement avoir un impact minimum et ne pas interférer. De plus, un NIDS doit tirer des leçons de son expérience afin de générer le moins de "faux négatifs" possibles. Il doit parallèlement posséder un moteur de filtrage fiable et performant remontant un faible taux de "faux positifs". Un système de filtrage des alarmes issus d'un NIDS est le coeur de ce moteur. Il n'est pas en réalité un système indépendant mais un module intégré au sein de ce système comme un post-processeur dont le but est de minimiser le pourcentage des "faux positifs". L'aspect évolutif d'un NIDS est alors indispensable pour le système de filtrage. Il doit aussi être évolutif et s'adapter parfaitement à la dynamique des architectures qu'il surveille et des comportements qu'il essaye de reconnaître.
Le fonctionnement de l'architecture de filtrage que nous avons brièvement présentée précédemment est basé sur les hypothèses suivantes :

1. La structure du réseau considéré (surveillé) est fixe, c.à.d il n'y a pas apparition de nouveaux équipements.
2. Le contexte permettant de reconnaître un comportement type n'évolue pas. En d'autre termes, il n'y a pas apparition de nouveaux types d'alertes.
3. Les comportements types eux-mêmes n'évoluent pas.

Afin que cette architecture soit dynamique et évolutive et par suite applicable en temps réel, il faut proposer des solutions permettant de lever ces hypothèses, en essayant de résoudre les trois problèmes de manière dynamique à partir d'un modèle de base. D'autre part, nous nous fixons comme contrainte de déclencher le réapprentissage de ce modèle seulement lorsque cela est nécessaire. Cette contrainte est facilement interprétable en terme d'utilisation de ressources : nous ne désirons pas consommer des ressources inutiles en fixant une fréquence de réapprentissage constante qui peut déclencher un apprentissage coûteux et inutile dans certains cas, ou l'effectuer trop tard dans d'autres situations. Il nous faut donc proposer une approche capable de déterminer l'instant où l'architecture n'est plus valide et il faut la mettre à jour. Cette approche doit être paramétrable et configurable suivant les préférences de l'administrateur. La décision de ré-apprendre ou de mettre à jour le système peut être basée sur différentes causes possibles, parmi lesquelles on peut citer l'évolution du réseau ou du NIDS et l'évolution des comportements types. Nous allons donc nous intéresser respectivement à ces deux situations.

3 Evolution du réseau ou du NIDS

3.1 Problème 1 : intégration de nouveaux équipements réseaux

Au cours du temps, de nouvelles $IP_{interne}$ (i.e., nouvelles machines, nouveaux équipements réseau routeurs, parefeux ou autres) sont ajoutées au réseau, d'autres sont enlevées et ainsi de suite. Le NIDS et le système de filtrage doivent être capables à se reconfigurer automatiquement de façon à prendre en compte les nouvelles modifications et les intégrer sans aucun arrêt ou perturbation.
Rappelons que notre architecture de filtrage est composée de deux phases principales [15] : *prétraitement* et *détection*. Dans la première phase, un nombre de comportements types à destination des $IP_{interne}$ est déterminé à partir des vecteurs resumés de chaque couple de machine en connection ($IP_{externe}, IP_{interne}$). Ces vecteurs résument les types d'alarmes générés par le système de détection d'intrusion durant une série des fenêtres temporelles glissantes. Ajouter ou supprimer des nouvelles $IP_{interne}$ ne perturbe donc pas cette phase.
La deuxième phase de classification est modulaire, et il est très facile de rajouter un nouveau module de classification locale pour les nouvelles $IP_{interne}$ et puis de les prendre ensuite en compte dans la classification globale. Par conséquent, l'intégration des nouvelles $IP_{interne}$ ne pose aucun problème et n'altère pas le fonctionnement de notre architecture.

3.2 Problème 2 : apparition de nouveaux types d'alertes

L'apparition des nouveaux types d'alertes est un événement très fréquent dans le domaine de la sécurité informatique. Très régulièrement, des nouvelles attaques sont découvertes et étudiées par les administrateurs de sécurité afin de mettre à jour les bases de signatures utilisées dans leurs NIDS. Alors, pour chaque nouvelle attaque, une nouvelle alerte significative de cette attaque est utilisée par les NIDS. Cet événement pose un grand défi sur le comportement de notre architecture. En effet, et comme mentionné dans [15], les comportements types sont créés par apprentissage d'une carte auto-organisatrice de Kohonen à partir d'une base de données dont les attributs sont les différents types d'alertes générées dans une fenêtre de temps. Ainsi, l'apparition d'un nouveau type d'alertes va modifier la dimension d'entrée du problème et entraîner un ré-

apprentissage de la carte. Les paramètres de cette nouvelle carte peuvent être initialisés de manière intelligente à partir de ceux de l'ancienne carte, en rajoutant autant de dimensions que de nouveaux types d'alertes. Les valeurs des anciennes composantes des vecteurs prototypes peuvent être initialisées aux valeurs de l'ancienne carte. Les valeurs des nouvelles composantes peuvent être initialisées de manière aléatoire ou uniforme, comme cela se passe dans le cas classique.

4 Evolution des comportements types

Le dernier problème à étudier concerne l'évolution éventuelle des comportements types modélisés par notre carte de Kohonen. Plusieurs cas sont envisageables lorsque de nouvelles données arrivent :

– La nouvelle observation est proche de l'un des K comportements-types modélisés par la carte. Dans ce cas, le système est encore valide et représentatif des nouvelles observations. Cependant, il y a possibilité de *glissement des comportements types* existants.
– La nouvelle observation est projetée dans une zone de l'espace située loin de toutes les comportements et doit donc être considérée comme invalide. Les observations invalides sont soient des données bruitées (qu'il faudrait rejeter) soient significatives de *nouveaux comportements types* qu'il faudrait maintenant détecter.

Nous proposons donc d'utiliser le modèle SOM pour détecter la présence de données invalides. Le processus de validation des données et de traitement des données invalides est illustré dans les deux premières parties de la figure 1 et décrit dans les sous-sections suivantes.

4.1 Modèle de base (SOM)

Nous avons à notre disposition une carte SOM composée d'un ensemble de K comportements types $(C_{i=1}^{K})$ de vecteurs prototypes $\{\omega_1, \ldots, \omega_K\}$ créés à partir d'un ensemble d'apprentissage $X = \{X_1, \ldots, X_N\}$. Chaque prototype ω_i est représentatif d'un comportement type C_i. L'espace de référence est ainsi divisé en K classes $(C_i)_{i=1}^{K}$. On peut voir la carte SOM comme une modélisation paramétrique de ces comportements types, chaque comportement étant modélisé par une fonction de densité gaussienne autour du vecteur prototype.

4.2 Décision de rejet et clustering des points rejetés

Décision avec rejet. les cartes topologiques de Kohonen ne sont pas seulement une méthode de visualisation et de classification des données de grande dimension. Elles peuvent également être utilisées pour détecter des données atypiques en contrôlant la distance entre chaque vecteur d'entrée x et le bmu (*best matching unit*, prototype le plus proche du point présenté). Cette technique peut être vue comme une variante du concept de rejet de distance décrit par exemple par [14]. Nous définissons "l'activation"

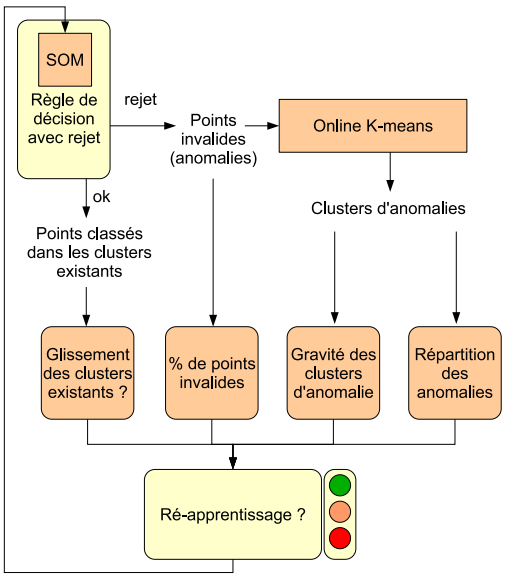

FIG. 1 – *Module de décision de ré-apprentissage.*

du prototype ω_i pour le vecteur d'entrée x en utilisant un noyau gaussien $k_i(x) = \exp(\frac{-1}{2\sigma_i^2} \parallel x - w_i \parallel^2)$, où σ_i est un paramètre qui définit la région d'influence du prototype i. σ_i peut être estimé par l'écart-type empirique des n vecteurs d'entrées activant le prototype i. Plus σ_i est grand, plus la zone d'influence de ω_i est grande et donc, plus l'activation $k_i(x)$ est proche de 1. Si l'activation $k_b(x)$ du bmu ω_b(i.e. le plus proche) est inférieure à un certain seuil α , le vecteur x est alors considéré comme invalide (à rejeter).

Ce procédé de rejet des données atypiques met en application une procédure de détection de nouveaux échantillons qu'il faudrait peut-être inclure dans l'ensemble d'apprentissage. Le rejet peut être dû à une limitation de l'ensemble d'apprentissage. Il est donc nécessaire de stocker ces vecteurs rejetés pour permettre une interprétation ultérieure par un utilisateur ou un éventuel ré-apprentissage du système.

Cette procédure de détection peut recevoir une interprétation probabiliste. Les vecteurs d'entrée x étant supposés avoir une distribution normale conditionnellement à C_i de moyenne ω_i et de variance $\sigma_i^2 I$ (I matrice identité). L'activation du prototype i par le vecteurs d'entrée x est donc proportionnelle à la densité de probabilité de x conditionnellement à la classe C_i. Cette procédure de rejet des données invalides permet d'éliminer les vecteurs x qui paraissent peu vraisemblables pour chacune de ces densités de classe (voir [21] page 25).

Définition du seuil α Nous avons à notre disposition l'ensemble de n_A vecteurs x de l'ensemble d'apprentissage. Nous allons considérer qu'il y a une probabilité p que ces données sont invalides ou aberrantes. Nous calculons donc l'activation $K_b(x)$ du bmu de ces n_A vecteurs en supposant que la distribution des données dans les différents

classes est normale de même variance σ^2. Nous considérons donc que $p\%$ des données de l'ensemble d'apprentissage ont une activation trop faible pour être considérées comme valides, comme le propose [25] pour une problématique proche. Il suffit donc de classer ces données par ordre croissant d'activation. Le seuil α correspond donc à l'activation $K_b(x)$ de rang $n_A * p/100$.

Clustering des points rejetés. Pour chaque nouveau vecteur projeté, on calcule l'activation de plus proche prototype. Si cette activation est plus petite que celle déterminée en 4.2, alors ce vecteur est considéré comme invalide. Les données rejetées peuvent être ou bien des bruits, ou bien d'éventuels nouveaux comportements. L'étape suivante consiste à essayer de regrouper ces données et étudier la nature des clusters obtenus que nous appellerons "clusters d'anomalie". Pour le regroupement des données nous avons utilisé l'algorithme *On-line K-means*, variante itérative de l'algorithme *K-means* [19]. Le nombre de clusters n'est pas fixé à l'avance mais déterminé en ligne suivant la nature des données.

5 Décision de ré-apprentissage

5.1 Reconnaître plusieurs situations

A l'issue de la phase précédente, identification et clustering des points invalides, les données atypiques ou invalides sont regroupées dans plusieurs clusters que nous nommerons *clusters d'anomalie*. La dernière phase consiste à étudier la nature de ces clusters d'anomalies et à prendre (si nécessaire) la décision de mise à jour du système. Cette décision est fonction de plusieurs paramètres. Le premier paramètre est la *dissimilarité* entre les nouvelles données et les comportements types existants. Si, au cours de temps, on remarque qu'un pourcentage *important* de données projetées est considéré comme *invalide*, alors les comportements types existants ne sont peut-être plus de bons représentants de ces nouvelles données.

Le deuxième paramètre intéressant est l'*importance* des clusters d'anomalie obtenus. Comme mentionné dans [15], nous avons classifié les alertes suivant trois niveaux de dangerosité. De plus, la nature de chaque comportement type est déterminée à partir des 5 alertes les plus importantes du vecteur prototype. Nous pouvons appliquer le même raisonnement sur les clusters d'anomalie pour avoir une idée de leur dangerosité. La présence de nouveaux clusters dangereux est évidemment un indicateur pour reconfigurer le système.

Le troisième paramètre à étudier est la *répartition des points* dans les clusters d'anomalie. Une accumulation des points dans un nombre limité de clusters d'anomalie est le signe d'apparition de nouveaux comportements types.

Pour finir, l'évolution des comportements-types modélisés par la carte de Kohonen peut se traduire par un *glissement* des données observées par rapport aux vecteurs prototypes existants. Cela est le signe que les anciens prototypes ne sont plus les bons représentants des données projetées et qu'il est indispensable de prendre ces nouvelles données en compte pour la détermination des nouveaux prototypes représentatifs des comportements-types.

En résumé, la décision de ré-apprendre la carte est fonction des quatre indicateurs suivants illustrés dans le bas de la figure 1 :

- pourcentage des points rejetés,
- gravité des clusters d'anomalie,
- répartition des points dans les clusters d'anomalie,
- glissement des comportements types existants.

Pourcentage des points rejetés. La question qui se pose maintenant est : à quel instant peut-on considérer la carte de Kohonen insuffisante et ré-apprendre le système ? En d'autres termes, pour quel pourcentage de données invalides est-il nécessaire de ré-apprendre le système ? Pour répondre à cette question, nous reformulons le problème sous la forme d'un test d'hypothèse statistique. La question sera alors quel est le pourcentage acceptable de points invalides dans la carte courante ? Ceci nous amène à faire *un test de comparaison de deux pourcentages*. Soient P_o le pourcentage observé de données invalides parmi les nouvelles données projetées et P_t le pourcentage théorique jugé correct. Dans le cas d'un test unilatéral à droite, les hypothèses sont donc $H_0 : P_o = P_t$ et $H_1 : P_o > P_t$.

Comme nous ne travaillons qu'avec des bases de grande taille, alors le test d'hypothèses pour un pourcentage repose sur les mêmes principes que le test d'hypothèses pour une moyenne et nous pouvons considérer que la distribution d'échantillonnage suit une *loi normale*. Soit z_α la valeur correspondant au seuil de signification désiré. Le rapport critique $R.C$ est exprimé par $R.C. = \frac{|P_o - P_t|}{\sigma_p}$ avec $\sigma_p = \frac{\sqrt{P_o(1-P_o)}}{\sqrt{n}}$ et n la taille de l'échantillon des nouvelles données projetées. Classiquement, si $R.C.$ est inférieur à z_α alors l'hypothèse H_0 est acceptée sinon elle est rejetée. Cette approche revient à calculer $P(R.C.)$ et à comparer cette valeur à α pour accepter ou non H_0. Dans le cadre de notre approche, nous utiliserons cette probabilité $P(R.C.)$ comme indicateur d'*invalidité* des nouvelles données : plus elle est grande et proche de 1 plus il peut être intéressant de ré-apprendre le modèle.

Gravité des clusters d'anomalies. Pour l'interprétation des clusters d'anomalies obtenus, rappelons que nous avons admis dans [15] que chaque vecteur prototype d'un cluster, représentant des vecteurs projetés dans ce cluster, peut être décrit par ses 5 variables les plus significatives (les 5 top variables, x_i $(i = 1, \ldots, 5)$). De plus, nous avons classé les alertes selon trois niveaux de sévérité $N = \{L, M, H\}$ avec *L=low, M=médium* et *H=high* où chaque niveau correspond à une pondération différente [15]. A partir de ces éléments, nous définissons la *gravité* d'un cluster par la somme (normalisée) des sévérités des 5 variables les plus représentatives du vecteur prototype de ce cluster : $Gr(Cluster) = \frac{\sum_{i=1}^{5} N(x_i)}{5.H}$.

Dans le cadre de notre approche, nous utiliserons cette grandeur comme indicateur de*gravité* des clusters d'ano-

malies : plus elle est grande et proche de 1 plus il peut être intéressant de ré-apprendre le modèle à cause de l'apparition d'une nouvelle attaque grave qui n'existait pas dans l'ensemble d'apprentissage.

Répartition des points dans les clusters d'anomalie. Après le regroupement des données invalides dans les clusters d'anomalie, l'étape suivante consiste à étudier la répartition des données dans ces clusters. Une répartition des points *non uniforme* est le signe qu'il y a cumul de points dans peu de clusters , ce qui peut être synonyme de l'apparition de nouveaux comportements types qu'il faudrait désormais prendre en compte. Comme pour les autres indicateurs, nous formulons ce problème sous forme d'un test d'hypothèse statistique avec les hypothèses H_0 : *répartition uniforme des points dans les clusters* et H_1 : *répartition non uniforme*.
Le test classiquement utilisé pour étudier ce genre de problème est le test de Khi2 (χ^2). Soit N l'ensemble des points considérés comme invalides par la décision de rejet. Cet ensemble constitue en effet l'échantillon sujet de ce test. Cet échantillon est réparti dans R classes d'anomalie distinctes $(A_1, \ldots, A_R)$. Soient $o_i (i = 1, \ldots, R)$ les effectifs observés et e_i les effectifs théoriques (i.e. ceux supposant une répartition uniforme). On calcule $Q = \sum_{i=1}^{R} \frac{(o_i - e_i)^2}{e_i}$. La statistique Q donne une mesure de l'écart existant entre les effectifs théoriques attendus et ceux observés dans l'échantillon. On compare ensuite cette valeur Q avec une valeur $\chi^2_{R-1,\alpha}$, où $R-1$ est le nombre de degrés de liberté et α est la tolérance. Si $Q > \chi^2_{R-1,\alpha}$ alors l'hypothèse H_0 est rejetée. Dans le cadre de notre approche, nous utiliserons cette probabilité $P(R.C.)$ comme indicateur de *répartition* des anomalies dans les clusters d'anomalie : plus elle est grande et proche de 1 plus il peut être intéressant de ré-apprendre le modèle à cause de l'apparition significative de nouveaux clusters qu'il faudrait prendre désormais en compte dans le modèle de base.

Glissement des comportements-types. En projetant de nouvelles données dans la carte, il peut arriver que le vecteur prototype de chaque comportement type ne soit plus un bon représentant des données projetées. Le nombre de comportements types peut rester le même mais leur "description" peut *évoluer* au cours de temps. Par conséquent, il est indispensable de prendre ces nouvelles données en compte pour la détermination des nouveaux centres ou prototypes. Rappelons que la distribution des points dans chaque comportement type est supposée suivre une loi gaussienne centrée au vecteur prototype et de variance supposée indépendante du comportement-type
Pour contrôler l'évolution des vecteurs prototypes et détecter s'il y a apparition d'échantillons issus d'une distribution différente, nous allons poser le problème sous forme d'un test statistique inspiré du test CUSUM [20, 7].
En effet, soit $\{X_1, \ldots, X_{n_A}\}$ l'ensemble des données d'apprentissage de taille n_A. Soient $E_i = dist(X_i, bmu_i)$ la distance de chaque vecteur à son *bmu* et le vecteur d'erreur $Err_A = \{E_1, \ldots, E_i, \ldots, E_{n_A}\}^t$. Supposons que l'échantillon Err_a suive une loi de paramètres μ_A et σ. L'idée ici est de comparer la distribution de Err_A à celle obtenue sur l'ensemble des données d'apprentissage et des nouvelles données obtenues depuis. Comme nous travaillons sur des échantillons de grande taille, alors le problème se reduit à un test de comparaison de moyenne de deux échantillons qui suivent une loi normale. Soit $Err_B = \{E_1, E_2, \ldots, E_{n_A}, E_{n_A+1}, \ldots, E_{n_A+k}\}$ le vecteur d'erreur cumulé et soient μ_A et μ_B les moyennes respectives des deux vecteurs aléatoires Err_A et Err_B. Pour simplifier le problème nous allons supposer que les deux vecteurs ont la même variance. Les hypothèses sont alors définies par $H_0 : \mu_A = \mu_B$ et $H_1 : \mu_A \neq \mu_B$.
La valeur du rapport critique $R.C$ est estimée à partir des données et comparée avec la valeur $z_{\alpha/2}$ correspondante de la table de Gauss. Si $R.C. > z_{\alpha/2}$, l'hypothèse nulle est rejetée. Dans le cadre de notre approche, nous utiliserons cette probabilité $P(R.C.)$ comme indicateur de *glissement* des clusters existants : plus elle est grande et proche de 1 plus il peut être intéressant de ré-apprendre le modèle.

5.2 Décision multi-critère

La dernière étape du processus de traitement est la phase de décision de ré-apprentissage de la carte de Kohonen. Cette décision de mettre à jour le système est fonction des quatre indicateurs précédemment étudiés. Nous proposons d'utiliser un *réseau bayésien* comme fonction de décision. Pour construire un réseau de ce type, il faut commencer par définir clairement les *variables* qui nous intéressent. La seconde étape consiste à établir le *graphe d'indépendance conditionnelle* entre les variables. Pour finir, il faut déterminer les *lois de probabilités conditionnelles* de chaque variable.

Définition des variables Nous possédons deux familles de variables : tout d'abord la variable de décision que nous appelerons FEU, variable discrète qui peut prendre trois valeurs : $Vert$ (le système actuel est encore valide), $Orange$ (incompatibilité avec les nouvelles données) et $Rouge$ (ré-apprentissage indispensable du système).
L'autre famille de variables est constituée des quatre indicateurs statistiques décrivant les situations possibles de ré-apprentissage :
- *Invalide* : probabilité que le pourcentage des données invalides dépasse le seuil accepté.
- *Répartition* : probabilité de répartition non-uniforme des points dans les clusters d'anomalie.
- *Gravité* : gravité maximale des clusters d'anomalie.
- *Glissement* : probabilité de glissement des prototypes des clusters existants.

Toutes ces variables sont *continues* avec des valeurs comprises entre 0 et 1.

Graphe d'indépendance Nous proposons d'utiliser un réseau bayésien naïf pour relier les variables (cf. figure 2). Ce type de réseau est simple et basé sur une hypothèse d'indépendance entre les variables sachant la variable classe.

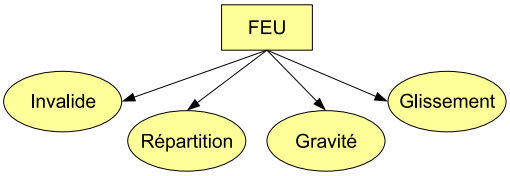

FIG. 2 – *Graphe d'indépendance du réseau bayésien naïf utilisé comme fonction de décision de réapprentissage.*

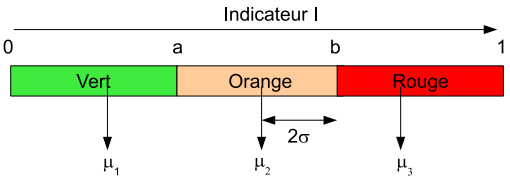

FIG. 3 – *Estimation des paramètres des densités $p(I|FEU)$ à l'aide d'un expert : les valeurs $\mu_1, \mu_2, \mu_3, \sigma$ sont définies à partir des bornes a et b données par l'expert pour chacun des indicateurs.*

Les probabilités conditionnelles Il reste à déterminer pour chaque noeud du graphe la distribution de probabilité conditionnelle $p(noeud|parent)$. Pour la variable FEU, cela revient à déterminer la probabilité que le système soit instable $p(FEU = Rouge)$ et ses complémentaires $p(FEU = Orange)$ et $p(FEU = Vert)$. Comme nous ne possédons aucune connaissance a priori du noeud FEU, nous utilisons une loi uniforme $P(FEU) = [1/3, 1/3, 1/3]$.

Pour les autres variables, il faut déterminer la densité de probabilité $p(I|FEU)$. La variable I est une variable générique qui représente chacun des quatre indicateurs statistiques. Nous décidons de déterminer ces valeurs à partir d'avis d'experts. Au lieu de demander directement à l'expert ces valeurs, nous proposons de lui demander d'indiquer les intervalles dans lesquels il pense que la variable FEU est égale à *Rouge*, *Orange* ou *Vert*.

Pour estimer ces probabilités, nous procédons de la manière suivante :

- Comme la variable I prend ses valeurs dans l'intervalle $[0; 1]$, alors nous décomposons cette intervalle en trois intervalles comme indiqué dans la figure 3. Les bornes de ces intervalles (i.e., a et b) sont des paramètres à définir par l'expert.
- $p(I|FEU = Vert)$ est une gaussienne de paramètres (μ_1, σ_1) avec $\mu_1 \in [0; a]$.
- $p(I|FEU = Orange)$ est une gaussienne de paramètres (μ_2, σ_2) avec $\mu_2 \in [a; b]$.
- $p(I|FEU = Rouge)$ est une gaussienne de paramètres (μ_3, σ_3) avec $\mu_3 \in [b; 1]$.
- L'écart-type est supposé identique pour les trois gaussiennes ($\sigma_1 = \sigma_2 = \sigma_3 = \sigma$).

Calcul des paramètres Les valeurs de μ_1, μ_2, μ_3 et σ sont maintenant estimées en fonction de a et b. Notons ici que les valeurs de a et b ne sont pas obligatoirement identiques pour les quatre indicateurs. Le calcul se fait ici d'une facon paramétrique. Commencons par le calcul de μ_2 : nous choisissons μ_2 comme milieu de $[a; b]$ comme indiqué dans la figure 3 ($\mu_2 = \frac{a+b}{2}$). En nous basant sur une propriété fondamentale de la loi normale :"*l'intervalle $[\mu - 2\sigma, \mu + 2\sigma]$ est la plage de normalité au niveau de confiance 95%*", alors nous choisissons $b - \mu_2 = 2\sigma$ (voir figure 3). Cela nous donne alors $\sigma = \frac{b-a}{4}$. De même $\mu_1 + 2\sigma = a$ donc $\mu_1 = \frac{3a-b}{2}$ et $\mu_3 = b + 2\sigma$, soit $\mu_3 = \frac{3b-a}{2}$.

6 Expérimentations et résultats

Cette section est consacrée à la validation séparée des indicateurs statistiques proposés et étudiés dans la section précédente, en nous concentrons ici, pour des raisons de place, sur les indicateurs *Pourcentage des données invalides*, *Répartition des anomalies* et *Glissement des clusters existants*. Nous choisissons de calculer l'intérêt de la décision de ré-apprendre la carte SOM pour chaque indicateur séparément en comparant l'erreur de quantification moyenne sur toute la carte s'il n'y avait pas de réapprentissage et celle calculée en prenant en compte les décisions de ré-apprentissage. La carte SOM avait été déterminée à partir d'une base d'apprentissage qui contient 41877 vecteurs caractéristiques distribués sur 800 fenêtres temporelles. La base de test utilisée pour les expériences suivantes contient 18491 vecteurs caractéristiques distribués sur 417 fenêtres. La phase de décision est *périodique* : pour chaque période fixée, les classes d'anomalie sont construites et les quatre indicateurs statistiques sont calculés. La période choisie ici est une *fenêtre temporelle* de 2 heures (la même que celle précédemment utilisée dans la première phase de notre architecture). En fonction des valeurs obtenues, la fonction de décision propose l'état du système ($FEU = Vert, Orange$, ou $Rouge$).

Pourcentage des données invalides La figure 4 présente les graphes obtenus lors de l'application de la règle de décision avec uniquement l'indicateur de pourcentage des données invalides. Pour chaque fenêtre temporelle, la probabilité que le pourcentage de données invalides dépasse le seuil (i.e. rejet de l'hypothèse nulle) est calculé. Le pourcentage théorique (seuil) est pris égal à $p_t = 5\%$. Les bornes $[a, b]$ utilisées dans l'estimation des densités de probabilités $P(I|FEU)$ pour cet indicateur sont $a = 30\%$ et $b = 50\%$.

La figure 4 (a) présente l'évolution du pourcentage des données invalides en fonction de temps. L'axe des x indique les fenêtres temporelles et l'axe des y le pourcentage des données invalides. Une fois que ce pourcentage dépasse le seuil, alors l'alarme rouge est déclenchée comme indiquée dans le graphe (c). Sur cette figure, le système n'est pas mis à jour pour chaque déclenchement de l'alarme rouge et le système continue sans ré-initialisation. Pour cette raison on voit dans le graphe (c) que l'alarme rouge est presque déclenchée à chaque instant. Le graphe

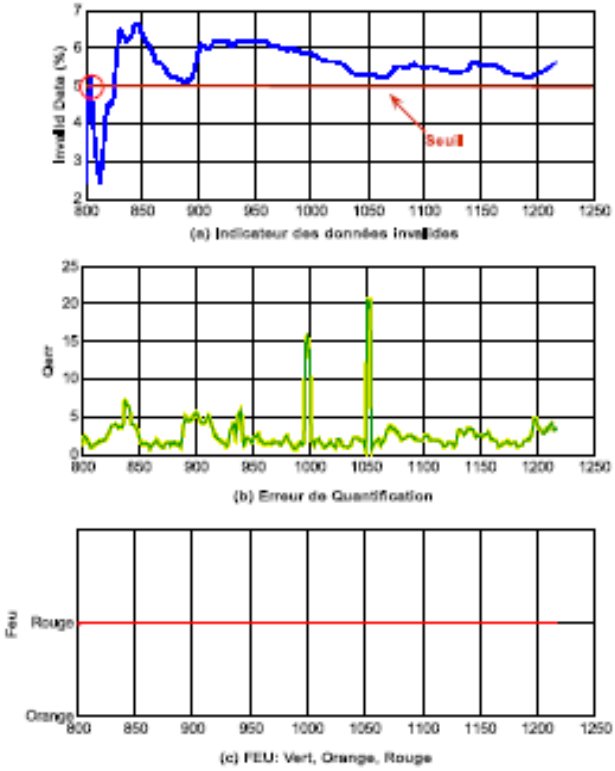

FIG. 4 – *La règle de décision appliquée à l'indicateur des données invalides (sans ré-apprentissage) : (a) pourcentage des points invalides, (b) erreur de quantification de la carte et (c) décision de ré-apprentissage. Dès le premier franchissement du seuil, indiqué en (a), le ré-apprentissage est fortement conseillée (feu rouge "allumé" en permanence).*

(b) présente l'erreur de quantification calculé en fonction de chaque fenêtre temporelle. L'erreur de quantification moyenne pendant cette période est égale à 2.75.
La figure 5 présente les mêmes graphes que la figure 4 avec une seule différence : *le système est ré-initialisé et mis à jour lors de chaque déclenchement de l'alarme rouge*. Comme le décrit la figure 5 (a), le système est réinitialisé trois fois : après la première alarme rouge $t = 810$, puis $t = 830$ et finalement $t = 931$. Les petits cercles indiquent l'instant où le système devient instable et l'alarme rouge est déclenchée. A la fin, nous obtenons une erreur de quantification moyenne égale à 2.6. La décision de ré-apprentissage à partir de cet indicateur nous permet bien de réduire le pourcentage des données invalides en adaptant régulièrement la carte de Kohonen tout en conservant une erreur de quantification moyenne proche.

Répartition des clusters d'anomalie Nous nous concentrons ici sur la probabilité de répartition non uniforme des données dans les clusters d'anomalie. Pour calculer les paramètres du réseau bayésien, les valeurs de a et b sont choisies égales à 40% et 97%.
Une répartition non uniforme entre les données regroupées

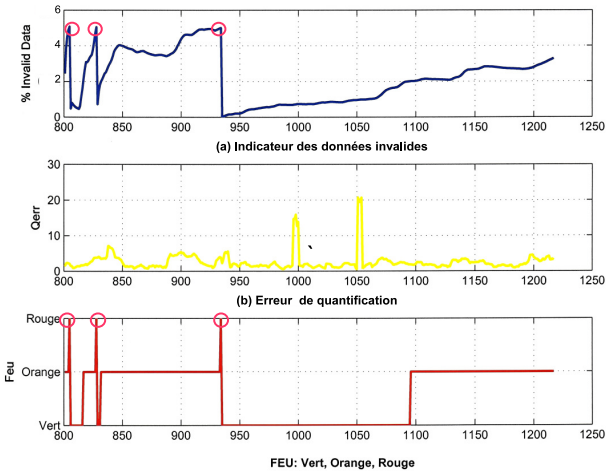

FIG. 5 – *La règle de décision appliquée à l'indicateur des données invalides (avec re-apprentissage).*

dans les clusters d'anomalie est le signe d'une accumulation de ces données dans quelques clusters. Ces grands clusters sont de nouveaux comportements types potentiels. Pour démontrer la capacité de l'architecture à détecter des nouveaux comportements types, nous décidons "d'annuler" virtuellement un cluster de la carte de Kohonen et de rejeter tous les points qui devraient normalement y être projetés. Cela va donc nous permettre d'étudier comment ces données maintenant rejetées vont être traitées dans la phase de détermination des clusters d'anomalie. L'application de l'algorithme *online K-means* aux points rejetés génère 5 clusters d'anomalie. 98% de ces points invalides sont regroupés dans le même cluster d'anomalie et les 2% restants sont dispersés dans les autres clusters. Cette répartition non uniforme des points d'anomalie dans les clusters déclenche alors une décision de ré-apprentissage. De plus, une étude plus précise, non détaillée ici faute de place, nous montre que le cluster d'anomalie "majoritaire" correspond effectivement au comportement type que nous avions annulé dans le modèle de base. Cet exemple nous permet d'illustrer que notre architecture a été capable de détecter un nouveau comportement type apparu dans les nouvelles données.

Glissement des clusters A chaque période de temps (fenêtre temporelle) et pour tester la validité des clusters existants, le test statistique adapté du test CUSUM est exécuté. Les bornes d'intervalle utilisées pour calculer les paramètres du réseau bayésien sont choisis égales à $[a = 40\%, b = 95\%]$. Les figures 6 et 7 présentent le comportement du système sur les données de tests. Dans la figure 6, la règle de décision a déclenché l'alarme rouge deux fois dans l'intervalle de temps compris entre les deux fenêtres temporelles $t = 900$ et $t = 950$ (figure (a)). Dans cet intervalle, et comme indiqué dans la figure (c), la probabilité a dépassé 90%. De même, la figure (b) indique que l'erreur de quantification atteint la valeur maximale dans cet intervalle.

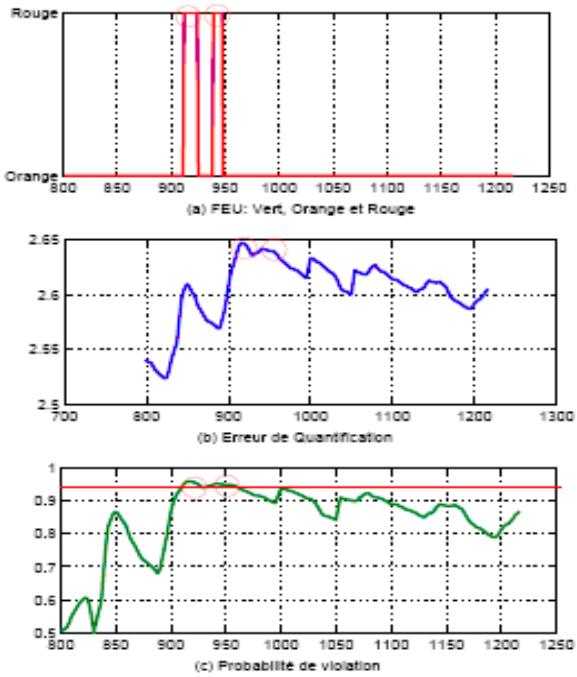

FIG. 6 – *Comportement du système suivant l'indicateur de glissement (sans ré-apprentissage) : (a) état de l'alarme, (b) erreur de quantification et (c) probabilité de glissement. L'indicateur ne dépasse le seuil qu'à deux instants, correspondants effectivement à une erreur de quantification élevée, ce qui déclenche une décision de ré-apprentissage (feu rouge)*

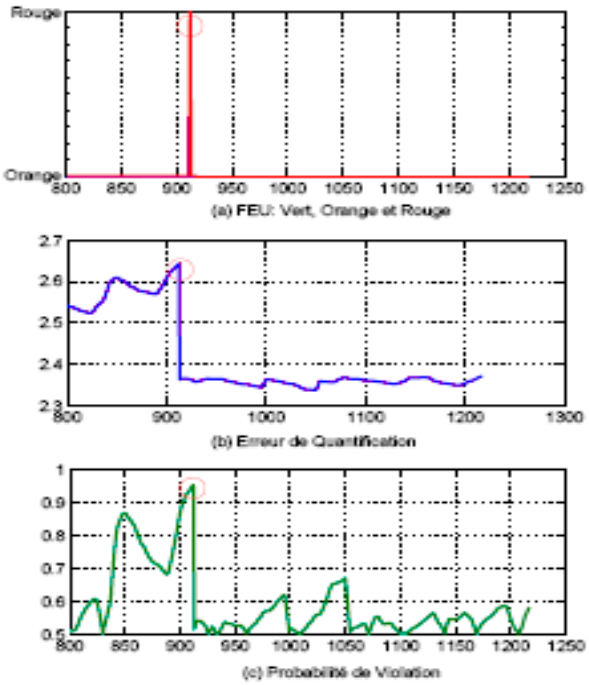

FIG. 7 – *Comportement du système suivant l'indicateur de glissement (avec ré-apprentissage) : (a) état de l'alarme, (b) erreur de quantification et (c) probabilité de glissement. L'indicateur ne dépasse le seuil qu'à un seul instant, correspondant effectivement à une erreur de quantification élevée, ce qui déclenche une décision de ré-apprentissage (feu rouge)*

Une deuxième expérience est exécutée sur les mêmes données, mais cette fois avec réapprentissage du système après chaque déclenchement de l'alarme rouge. La figure 7 (a,b et c) illustre le comportement du système. Après le premier déclenchement de l'alarme rouge, le système est réinitialisé et la carte de Kohonen est reconfigurée avec les nouvelles données projetées. Remarquons que le système ne déclenche plus la deuxième alarme rouge comme dans la première expérience (figure 7 (a)). La figure (b) montre l'évolution de l'erreur de quantification. Cette erreur moyenne passe de 2.7 avant le ré-apprentissage à 2.3 après le ré-apprentissage du système. Comme précédemment, la décision de ré-apprentissage à partir de cet indicateur nous permet bien de détecter le glissement éventuel des comportements types en adaptant régulièrement la carte de Kohonen tout en conservant une erreur de quantification moyenne proche.

7 Conclusion et perspectives

Nous avons traité dans cet article trois problèmes que l'on rencontre régulièrement lors de l'application en temps réel d'une architecture de filtrage ou de clustering : (a) l'évolution du réseau surveillé, (b) l'apparition de nouveaux types d'alertes, et (c) l'évolution des comportements-types de la carte de Kohonen. De part l'aspect modulaire de notre architecture, le premier problème ne pose aucun problème. Le deuxième problème oblige le ré-apprentissage du système car l'apparition d'un nouveau type d'alerte provoque un changement du nombre d'attributs du vecteur caractéristique utilisé pour créer les comportements types. Le troisième problème nous a amené à proposer quatre indicateurs statistiques et une fonction de décision permettant de déterminer l'état actuel du système (i.e., stable ou instable). Pour finir, la validité de ces indicateurs a été testée lors d'une phase d'expériences. Les résultats obtenus montrent que ces indicateurs donnent à l'administrateur une bonne idée sur l'évolution du système au cours du temps en précisant les moments critiques où il faudrait que le système se reconfigure.

Une de nos perspectives est maintenant "d'observer" l'administrateur système qui suit ou non les préconisations du module de ré-apprentissage. Cette observation permettra alors de prendre en compte les préférences de l'administrateur et de modifier en conséquence les paramètres de la fonction de ré-apprentissage.

De manière plus générale, nous avons abordé dans cet article le problème du ré-apprentissage, avec pour l'instant une démarche très spécifique à la détection d'intrusions et au choix du modèle de référence. Une perspective est de réfléchir à la généralisation de l'approche, avec d'un côté la définition d'indicateurs indépendants de l'application (ce qui est presque le cas de ceux que nous proposons), et sur-

tout d'indicateurs qui soient indépendants du modèle de base (ici la carte de Kohonen) ou liés à d'autres modèles de mélanges comme les graphes génératifs gaussiens utilisés par [16]. Nous allons aussi nous pencher sur la robustesse de la méthode par rapport aux hypothèses faites, avec par exemple l'hypothèse de variance identique pour tous les clusters. Cette hypothèse classique, peut-être utile pour la formalisation du problème, mais pas nécessaire dans la réalité. Nous en avons un premier exemple dans l'étude expérimentale menée dans cet article puisque nos données réelles ne vérifient pas l'hypothèse. Une étude de robustesse plus poussée doit maintenant être menée pour pouvoir généraliser la méthodologie proposée dans cet article.

Références

[1] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical report, Carnegie Mellon University, January 2000.

[2] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In *Network and Distributed System Security Symposium (NDSS 2000)*, pages 157–170, 2000.

[3] N. Ben Amor, S. Benferhat, and Z. Elouedi. Naive bayes vs decision trees in intrusion detection systems. In *ACM SAC 2004*, 2004.

[4] R. Bace. *Intrusion Detection*. Macmillan Technical Publishing, 2000.

[5] S. Benferhat and K. Tabia. Systèmes de détection d'intrusions hybrides, partie i : intégration d'une approche comportementale. In *5th Conference on Security and Network Architectures (SAR 2006) and 3rd Conference on Security in Information Systems (SSI'2006)*, 2006.

[6] Y. Bouzida. *Application de l'analyse en composante principale pour la détection d'intrusions et détection de nouvelles attaques par apprentissage supervisé*. PhD thesis, GET/ENST Bretagne, Mars 2006.

[7] F.J. Bryan, I. Darryl, and Mackenzie. Cusum environmental monitoring in time and space. *Environmental and Ecological Statistics*, 10 :231–247, 2003.

[8] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *17th Annual Computer Security Applications Conference (ACSAC)*, pages 22–31, 2001.

[9] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, pages 202–215, 2002.

[10] O. M. Dain and R. K. Cunningham. Building scenarios from a heterogeneous alert stream. In *IEEE SMC Information Assurance Workshop*, 2001.

[11] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion detection systems. *Annales des Télécommunications*, 55(7-8) :361–378, 2000.

[12] H. Debar, B. Morin, F. Cuppens, F. Autrel, L. Mé, B. Vivinis, S. Benferhat, M. Ducassé, and R. Ortalo. Détection d'intrusions : Corrélation d'alertes. *Revue Technique et Science Informatique (TSI)*, 23 :359–390, 2004.

[13] H. Debar and A. Wespi. Aggregation and correlation of intrusion alerts. In *4th Workshop on Recent Advances in Intrusion Detection (RAID 2001)*, pages 85–103, 2001.

[14] B. Dubuisson and M. Masson. A statistical decision rule with incomplete knowledge about classes. *Pattern Recognition*, 26(1) :155–165, 1993.

[15] A. Faour, P. Leray, and B. Eter. A SOM and bayesian network architecture for alert filtering in network intrusion detection systems. In *2nd IEEE International Conference On Information and Communication Technologies : From Theory to Applications (ICTTA 2006)*, pages 1161–1166, Damascus, Syria, 2006.

[16] P. Gaillard, M. Aupetit, and G. Govaert. Apprentissage statistique de la topologie d'un ensemble de données étiquetées. In *7èmes journées d'Extraction et de Gestion des Connaissances (EGC)*, pages 455–460, 2007.

[17] T. Kohonen. *Self Organizing Maps*. Springer Verlag, 1995.

[18] P. Leray. Réseaux bayésiens : apprentissage et modélisation de systèmes complexes. Habilitation à diriger les recherches, Université de Rouen, Novembre 2006.

[19] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematics, Statistics and Probability*, volume 1, pages 281–298, 1967.

[20] E. S. Page. Cumulative sum control charts. *Technometrics*, 3 :1–9, 1961.

[21] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[22] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A high-performance network intrusion detection system. In *6th ACM Conference on Computer and Communications Security*, pages 8–17, 1999.

[23] S. Staniford, J.A. Hoagland, and J.M. McAlernay. Practical automated detection of stealthy portscans. In *ACM Computer and Communications Security IDS Workshop*, pages 1–7, 2000.

[24] A. Valdes and K. Skinner. Probabilistic alert correlation. In *4th Workshop on Recent Advances in Intrusion Detection (RAID)*, pages 54–86, Berlin, 2001. LNCS. Springer Verlag.

[25] N. Valentin. *Construction d'un capteur logiciel pour le contrôle automatique du procédé de coagulation en traitement d'eau potable*. PhD thesis, UTC, 2000.

[26] D. Zamboni. *Using internal sensors for computer intrusion detection*. PhD thesis, Purdue university, 2001.