Online Adaboost-Based Parameterized Methods for Dynamic Distributed Network Intrusion Detection

Weiming Hu, Jun Gao, Yanguo Wang, and Ou Wu
(National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190)
{wmhu, jgao, ygwang, wuou}@nlpr.ia.ac.cn

Stephen Maybank
(Department of Computer Science and Information Systems, Birkbeck College, Malet Street, London WC1E 7HX)
sjmaybank@dcs.bbk.ac.uk

Abstract: Current network intrusion detection systems (NIDS) lack the adaptability to the frequently changing network environments. Furthermore, intrusion detection in the new distributed architectures is now a major requirement. In this paper, we propose two online Adaboost-based intrusion detection algorithms. In the first algorithm, a traditional online Adaboost process is used where decision stumps are used as weak classifiers. In the second algorithm, an improved online Adaboost process is proposed, and online GMMs are used as weak classifiers. We further propose a distributed intrusion detection framework, in which a local parameterized detection model is constructed in each node using the online Adaboost algorithm. A global detection model is constructed in each node by combining the local parametric models using a small number of samples in the node. This combination is achieved using an algorithm based on particle swarm optimization (PSO) and support vector machines (SVM). The global model in each node is used to detect intrusions. Experimental results show that the improved online Adaboost process with GMMs obtains a higher detection rate and a lower false alarm rate than the traditional online Adaboost process which uses decision stumps. Both the algorithms outperform existing intrusion detection algorithms. It is also shown that our PSO and SVM-based algorithm effectively combines the local detection models into the global model in each node: the global model in a node can handle the intrusion types which are found in other nodes, without sharing the samples of these intrusion types.

Index terms: Network intrusions, Dynamic distributed detection, Online Adaboost learning, Parameterized model

1. Introduction

Network attack detection is one of the most important problems in network information security. Currently there are mainly firewall, NIDS/NIPS (network-based intrusion detection and prevention systems) and UTM (unified threat management) like devices to detect attacks in network infrastructure. NIDS/NIPS detect and prevent network behaviors which violate or endanger network security. Basically, firewalls are used to block certain types of traffic to improve the security. NIDS and firewalls can be linked to block network attacks. UTM devices combine firewall, NIDS/NIPS and other capabilities onto a single device to detect similar events as standalone firewalls and NIDS/NIPS devices. Deep Packet Inspection (DPI) [52] adds analysis on the application layer, and then recognizes various applications and their contents. DPI can incorporate NIDS into firewalls. It can increase the accuracy of intrusion detection, but it is more time-consuming, in contrast to traditional package header analysis. This paper focuses on investigation of NIDS.

The current practical solutions for NIDS used in industry are misuse-based methods which utilize signatures

of attacks to detect intrusions by modeling each type of attack. As typical misuse detection methods, pattern matching methods search packages for the attack features by utilizing protocol rules and string matching. Pattern matching methods can effectively detect the well-known intrusions. But they rely on the timely generation of attack signatures, and fail to detect novel and unknown attacks. In the case of rapid proliferation of novel and unknown attacks, any defense based on signatures of known attacks becomes impossible. Moreover, the increasing diversity of attacks obstructs modeling signatures.

Machine learning deals with automatically inferring and generalizing dependencies from data to allow extrapolation of dependencies to unseen data. Machine learning methods for intrusion detection model both attack data and normal network data, and allow for detection of unknown attacks using the network features [60]. This paper focuses on machine learning-based NIDS. The machine learning-based intrusion detection methods can be classified as statistics-based, data mining-based, and classification-based. All the three classes of methods first extract low level features and then learn rules or models which are used to detect intrusions. A brief review of each class of methods is given below.

1) Statistics-based methods construct statistical models of network connections to determine whether a new connection is an attack. For instance, Denning [1] construct statistical profiles for normal behaviors. The profiles are used to detect anomalous behaviors which are treated as attacks. Caberera *et al*. [2] adopt the Kolmogorov-Smirnov test to compare observation network signals with normal behavior signals, assuming that the number of observed events in a time segment obeys the Poisson distribution. Li and Manikopoulos [22] extract several representative parameters of network flows, and model these parameters using a hyperbolic distribution. Peng *et al*. [23] use a nonparametric cumulative sum algorithm to analyze the statistics of network data, and further detect anomalies on the network.

2) Data mining-based methods mine rules which are used to determine whether a new connection is an attack. For instance, Lee *et al*. [3] characterize normal network behaviors using association rules and frequent episode rules [24]. Deviations from these rules indicate intrusions on the network. Zhang *et al*. [40] use the random forest algorithm to automatically build patterns of attacks. Otey *et al*. [4] propose an algorithm for mining frequent itemsets (groups of attribute value pairs) to combine categorical and continuous attributes of data. The algorithm is extended to handle dynamic and streaming data sets. Zanero and Savaresi [25] first use unsupervised clustering to reduce the network packet payload to a tractable size, and then a traditional anomaly detection algorithm is applied to intrusion detection. Mabu *et al*. [49] detect intrusions by mining fuzzy class association rules using genetic network programming. Panigrahi and Sural [51] detect intrusions using fuzzy logic, which combines evidence from a user's current and past behaviors.

3) Classification-based methods construct a classifier which is used to classify new connections as either attacks or normal connections. For instance, Mukkamala *et al*. [30] use the support vector machine (SVM) to distinguish between normal network behaviors and attacks, and further identify important features for intrusion detection. Mill and Inoue [31] propose the TreeSVM and ArraySVM algorithms for reducing the inefficiencies which arise when a sequential minimal optimization algorithm for intrusion detection is learnt from a large set of training data. Zhang and Shen [7] use SVMs to implement online intrusion detection. Kayacik *et al*. [5] propose

an algorithm for intrusion detection based on the Kohonen self organizing feature map (SOM). Specific attention is given to a direct labeling of SOM nodes with the connection type. Bivens *et al*. [26] propose an intrusion detection method in which SOMs are used for data clustering and multi-layer perceptron (MLP) neural networks are used for detection. Hierarchical neural networks [28], evolutionary neural networks [29], and MLP neural networks [27], have been applied to distinguish between attacks and normal network behaviors. Hu and Heywood [6] combine SVM with SOM to detect network intrusions. Khor *et al*. [55] propose a dichotomization algorithm in which rare categories are separated from the training data and cascaded classifiers are trained to handle both the rare categories and other categories. Xian *et al*. [32] combine the fuzzy K-means algorithm with the clonal selection algorithm to detect intrusions. Jiang *et al*. [33] use an incremental version of the K-means algorithm to detect intrusions. Hoglund *et al*. [34] extract features describing network behaviors from audit data, and use the SOM to detect intrusions. Sarasamma *et al*. [18] propose a hierarchical SOM which selects different feature subset combinations that are used in different layers of the SOM to detect intrusions. Song *et al*. [35] propose a hierarchical random subset selection-dynamic subset selection (RSS-DSS) algorithm to detect intrusions. Lee *et al*. [8] propose an adaptive intrusion detection algorithm which combines the adaptive resonance theory with the concept vector and the Mecer-Kernel. Jirapummin *et al*. [21] employ a hybrid neural network model to detect TCP SYN flooding and port scan attacks. Eskin *et al*. [20] use a *k*-nearest neighbor (*k*-NN)-based algorithm and a SVM-based algorithm to detect anomalies.

Although there is much work on intrusion detection, several issues are still open and require further research, for example:

- Network environments and the intrusion training data change rapidly over time, as new types of attack emerge. In addition, the size of the training data increases over time and can become very large. Most existing algorithms for training intrusion detectors are offline. The intrusion detector must be retrained periodically in batch mode in order to keep up with the changes in the network. This retraining is time consuming. Online training is more suitable for dynamic intrusion detectors. New data are used to update the detector and are then discarded. The key issue in online training is to maintain the accuracy of the intrusion detector.
- There are various types of attributes for network connection data, including both categorical and continuous ones, and the value ranges for different attributes differ greatly — from {0, 1} to describe the normal or error status of a connection, to [0, 107] to specify the number of data bytes sent from source to destination. The combination of data with different attributes without loss of information is crucial to maintain the accuracy of intrusion detectors.
- In traditional centralized intrusion detection, in which all the network data are sent to a central site for processing, the raw data communications occupy considerable network bandwidth. There is a computational burden in the central site and the privacy of the data obtained from the local nodes cannot be protected. Distributed detection [36], which shares local intrusion detection models learned in local nodes, can reduce data communications, distribute the computational burden, and protect privacy. Otey *et al*. [4] construct a novel distributed algorithm for detecting outliers (including network intrusions). Its

limitation is that many raw network data still need to be shared among distributed nodes. There is a requirement for a distributed intrusion detection algorithm to make only a small number of communications between local nodes.

In this paper, we address the above challenges, and propose a classification-based framework for the dynamic distributed network intrusion detection using the online Adaboost algorithm [58]. The Adaboost algorithm is one of the most popular machine learning algorithms. Its theoretical basis is sound, and its implementation is simple. Moreover, the AdaBoost algorithm corrects the misclassifications made by weak classifiers and it is less susceptible to over-fitting than most learning algorithms. Recognition performances of the Adaboost-based classifiers are generally encouraging. In our framework, a hybrid of online weak classifiers and an online Adaboost process results in a parameterized local model at each node for intrusion detection. The parametric models for all the nodes are combined into a global intrusion detector in each node using a small number of samples, and the combination is achieved using an algorithm based on particle swarm optimization (PSO) and SVMs. The global model in a node can handle the attack types which are found in other nodes, without sharing the samples of these attack types. Our framework is original in the following ways:

- In the Adaboost classifier, the weak classifiers are constructed for each individual feature component, both for continuous and categorical ones, in such a way that the relations between these features can be naturally handled, without any forced conversions between continuous features and categorical features.
- New algorithms are designed for local intrusion detection. The traditional online Adaboost process and a newly proposed online Adaboost process are applied, to construct local intrusion detectors. The weak classifiers used by the traditional Adaboost process are decision stumps. The new Adaboost process uses online Gaussian Mixture Models (GMM) as weak classifiers. In both cases the local intrusion detectors can be updated online. The parameters in the weak classifiers and the strong classifier construct a parametric local model.
- The local parametric models for intrusion detection are shared between the nodes of the network. The volume of communications is very small and it is not necessary to share the private raw data from which the local models are learnt.
- We propose a PSO and SVM-based algorithm for combining the local models into a global detector in each node. The global detector which obtains information from other nodes obtains more accurate detection results than the local detector.

The remainder of this paper is organized as follows. Section 2 introduces the distributed intrusion detection framework. Section 3 describes the online Adaboost-based local intrusion detection models. Section 4 presents the method for constructing the global detection models. Section 5 shows the experimental results. Section 6 summarizes the paper.

2. Overview of Our Framework

In the distributed intrusion detection framework, each node independently constructs its own local intrusion

detection model according to its own data. By combining all the local models**,** at each node, a global model is trained using a small number of the samples in the node, without sharing any of the original training data between nodes. The global model is used to detect intrusions at the node. Fig. 1 gives an overview of our framework which consists of the modules of data preprocessing, local models, and global models.

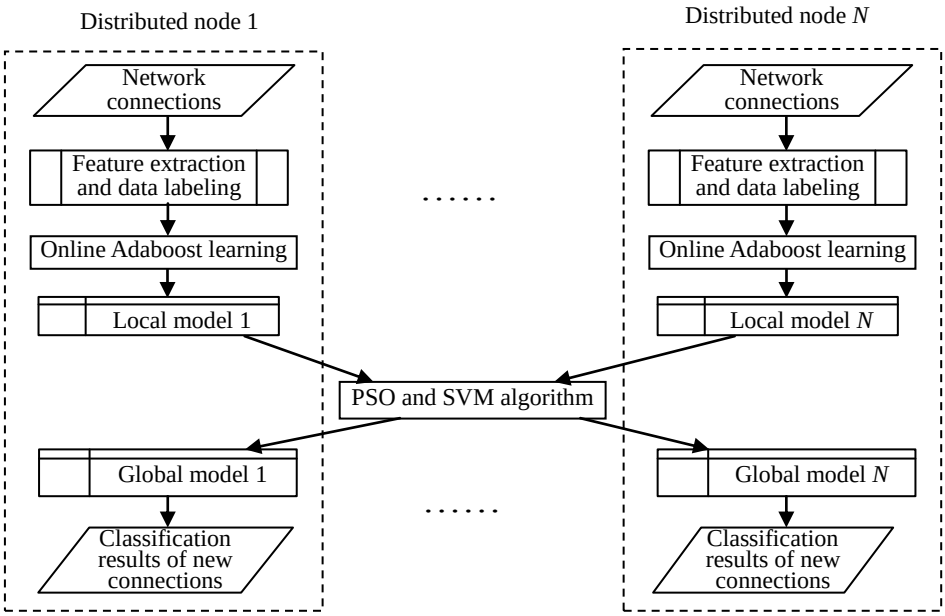


Fig. 1. Overview of the intrusion detection framework

1) Data preprocessing: For each network connection, three groups of features [9] which are commonly used for intrusion detection are extracted: basic features of individual TCP (transmission control protocol) connections, content features within a connection suggested by domain knowledge, and traffic features computed using a two-second time window [14]. The extracted feature values from a network connection form a vector $x = (x1, x2, \ldots, x_D)$, where D is the number of feature components. There are continuous and categorical features, and the value ranges of the features may differ greatly from each other. The framework for constructing these features can be found in [9]. A set of data **is** labeled for training purposes. There are many types of attacks on the Internet. The attack samples are labeled as "-1", "-2", ... depending on the attack type, and the normal samples are all labeled as "+1".

2) Local models: The construction of a local detection model at each node includes the design of weak classifiers and Adaboost-based training. Each individual feature component corresponds to a weak classifier. In this way, the mixed attribute data for the network connections can be handled naturally, and full use can be made of the information in each feature. The Adaboost training is implemented using only the local training samples at each node. After training, each node contains a parametric model which consists of the parameters of the weak classifiers and the ensemble weights.

3) Global models: By sharing all the local parametric models, a global model is constructed using the PSO and SVM-based algorithm in each node. The global model in each node fuses the information learned from all the local nodes using a small number of training samples in the node. Feature vectors of new network connections to the node are input to the global classifier, and classified as either normal or attacks. The results of the global model in the node are used to update the local model in the node and the updated model is then shared

by other nodes.

3. Local Detection Models

The classical Adaboost algorithm [37] carries out the training task in batch mode. A number of weak classifiers are constructed using a training set. Weights, which indicate the importance of the training samples, are derived from the classification errors of the weak classifiers. The final strong classifier is an ensemble of weak classifiers. The classification error of the final strong classifier converges to 0. However, the Adaboost algorithm based on offline learning is not suitable for networks. We apply online versions of Adaboost to construct the local intrusion detection models. It is proved in [38] that the strong classifier obtained by the online Adaboost converges to the strong classifier obtained by the offline Adaboost as the number of training samples increases.

In the following, we first introduce the weak classifiers for intrusion detection, and then describe the online Adaboost-based intrusion detection algorithms.

3.1. Weak classifiers

Weak classifiers which can be updated online match the requirement of dynamic intrusion detection. We consider two types of weak classifier. The first type consists of decision stumps for classifying attacks and normal behaviors. The second type consists of online GMMs which model a distribution of values of each feature component for each attack type.

3.1.1. Decision stumps

A decision stump is a decision tree with a root node and two leaf nodes. A decision stump is constructed for each feature component of the network connection data.

For a categorical feature *f*, the set of attribute values C^f is divided into two subsets C_i^f and C_n^f with no intersection, and the decision stump takes the form:

$$h_f(x)=\begin{cases}1 & x_f \in C_n^f \\ -1 & x_f \in C_i^f\end{cases} \tag{1}$$

where x_f is the attribute value of *x* on the feature *f*. The subsets C_i^f and C_n^f are determined using the training samples: for an attribute value *z* on a feature *f*, all the training samples whose attribute values on *f* are equal to *z* are found; if the number of attack samples in these samples is more than the number of normal samples, then *z* is assigned to C_i^f, otherwise, *z* is assigned to C_n^f. In this way, the false alarm rate for the training samples is minimized.

For a continuous feature *f*, the range of attribute values is split by a threshold *v*, and the decision stump takes the form:

$$h_f(x)=\begin{cases}1 & x_f > v \\ -1 & x_f \le v\end{cases}. \tag{2}$$

The threshold *v* is determined by minimizing the false alarm rate for the training samples.

The above design of weak classifiers for intrusion detection has the following advantages:

- The decision stumps operate on individual feature components. This first deals naturally with combination of information from categorical attributes and from continuous attributes, and second deals with the large variations in value ranges for the different feature components.
- There is only one comparison operation in a decision stump. The computation complexity for constructing the decision stumps is very low, and online updating of decision stumps can be easily implemented when new training samples are obtained**.**

The limitation of the above design of weak classifiers is that the decision stumps do not take into account the different types of attacks. This may influence the performance of the Adaboost method.

3.1.2. Online GMM

For the samples of each attack type or the normal samples, we use a GMM to model the data on each feature component. Let $c \in \{+1, -1, -2, \cdots, -M\}$ be a sample label, where "+1" represents the normal samples and "-1, -2,…-*M*" represents different types of attacks where *M* is the number of attack types. The GMM model θ_j^c on the *j*th feature component for the samples with label *c* is represented as:

$$\theta_j^c = \left\{\omega_j^c(i), \mu_j^c(i), \sigma_j^c(i)\right\}_{i=1}^{K} \tag{3}$$

where *K* is the number of GMM components indexed by *i*, and ω, μ and σ represent the weight, mean and standard deviation for the corresponding component. Then, the weak classifier on the *j*th feature is constructed as:

$$h_j(x) = \begin{cases} 1 & if \quad p(x \mid \theta_j^{+1}) > \dfrac{1}{M}\, p(x \mid \theta_j^c) \quad for \quad c = -1, -2, ..., -M \\ -1 & otherwise \end{cases} \tag{4}$$

where "1/*M*" is used to weight the probabilities for attack types in order to balance the importance of the attack samples and the normal samples. In this way, the sum of the weights for all the types of attack samples is equal to the weight of normal samples, and then the false alarm rate is reduced for the final ensemble classifier.

The traditional way for estimating the parameter vectors θ_j^c, using the *K*-means clustering and the expectation-maximization (EM) algorithm [41], from the training samples, is time-consuming and is not suitable for online learning. In this paper, we use the online EM-based algorithm [44] to estimate these parameters. Given a new sample (x, y) where *x* is the feature vector of the sample and $y \in \{1, -1, -2 \cdots\}$, the parameter vectors θ_j^y are updated using the following steps:

Step 1: Update the number $N_j^y(i)$ of the training samples belonging to the *i*th component of the GMM by:

$$\Gamma_i(x) = \begin{cases} 1 & if \quad \left| \dfrac{x - \mu_j^y(i)}{\sigma_j^y(i)} \right| \le T \\ 0 & else \end{cases} \tag{5}$$

$$N_j^y(i) \leftarrow N_j^y(i) + \Gamma_i(x) \tag{6}$$

where the binary function $\Gamma_i(x)$ determines whether (x, y) belongs to the ith component, and the threshold T depends on confidence limits required.

Update $\omega_j^y(i)$ by:

$$\omega_j^y(i) = \frac{N_j^y(i)}{\sum_{k=1}^{K} N_j^y(k)}. \tag{7}$$

Step 2: Calculate the following equation:

$$\mathbb{Z} = \sum_{i=1}^{K} \omega_j^y(i)\, p(x \mid \mu_j^y(i), \sigma_j^y(i))\Gamma_i(x). \tag{8}$$

where $\mathbb{Z}$ describes the relation between (x, y) and the GMM of θ_j^y.

If $\mathbb{Z} > 0$, then go to Step 3; else ($\mathbb{Z} = 0$ means that there is no relation between (x, y) and the GMM of θ_j^y), go to Step 5 for setting a new Gaussian component for (x, y).

Step 3: Update the sum $A_j^y(i)$ of the weighted probabilities for all the input training samples belonging to the ith component by:

$$\delta(i) = \omega_j^y(i)\, p(x \mid \mu_j^y(i), \sigma_j^y(i))\Gamma_i(x) / \mathbb{Z} \tag{9}$$

$$A_j^y(i) \leftarrow A_j^y(i) + \delta(i) \tag{10}$$

where $\delta(i)$ is the probability that (x, y) belongs to the ith component of the GMM.

Step 4: Update $\mu_j^y(i)$ and $\sigma_j^y(i)$ by:

$$\mu_j^y(i) \leftarrow \frac{A_j^y(i) - \delta(i)}{A_j^y(i)} \mu_j^y(i) + \frac{\delta(i)}{A_j^y(i)} x \tag{11}$$

$$\sigma^{2}{}_j^y(i) \leftarrow \frac{A_j^y(i) - \delta(i)}{A_j^y(i)} \sigma^{2}{}_j^y(i) + \frac{(A_j^y(i) - \delta(i))\delta(i)}{A_j^y(i)} (x - \mu_j^y(i))^2. \tag{12}$$

Exit.

Step 5: Reset a Gaussian component t in θ_j^y by:

$$t = \arg\min_i(\omega_j^y(i)) \tag{13}$$

$$\begin{cases} N_j^y(t) \leftarrow N_j^y(t) + 1 \\ A_j^y(t) \leftarrow A_j^y(t) + 1 \end{cases} \tag{14}$$

$$\begin{cases} \mu_j^y(t) = x \\ \sigma_j^y(t) = \sigma \end{cases}. \tag{15}$$

where σ is used to initialize the standard deviation of the component t.

The terms $\delta(i)/A_j^y(i)$ in (11) and $(A_j^y(i)-\delta(i))\delta(i)/A_j^y(i)$ in (12) are the learning rates of the algorithm. They are changed dynamically: getting smaller with the growth of the training samples. This behavior of the learning rates ensures that the online EM algorithm achieves the balance between the stability needed to keep the characteristics of the model learned from the previous samples and necessity of adapting the model as new samples are received.

The computational complexity of the online GMM for one sample is $O(K)$. While the online GMM, which can be used to model distributions of each type of attack, has a higher complexity than decision stumps, the complexity for the online GMM is still very low as *K* is a small integer. So, the online GMM is suitable in online environments.

3.2. Online ensemble learning

To adapt to online training in which each training sample is used only once for learning the strong classifier, online Adboost makes the following changes to the offline Adaboost:

- All the weak classifiers are initialized.
- Once a sample is input, a number of weak classifiers are updated using this sample, and the weight of this sample is changed according to certain criteria such as the weighted false classification rate of the weak classifiers.

In the following, we first apply the traditional online Adaboost algorithm to intrusion detection, and then we propose a new and more effective online Adaboost algorithm.

3.2.1. Traditional online Adaboost

Oza [38] proposes an online version of Adaboost, and gives a convergence proof. This online Adaboost algorithm has been applied to the computer vision field [42]. When a new sample is input to the online Adaboost algorithm, all the weak classifiers (ensemble members) are updated in the same order [10]. Let $\{h_t\}_{t=1,\ldots,D}$ be the weak classifiers. This online Adaboost algorithm is outlined as follows:

Step 1: Initialize the weighted counters λ_t^{sc} and λ_t^{sw} of "historical" correct and wrong classification results for the *t*-th weak classifier h_t as: $\lambda_t^{sc}=0$, $\lambda_t^{sw}=0$ (*t*=1, 2, …, *D*).

Step 2: For each new sample (*x*, *y*)

Initialize the weight λ of the current sample as: $\lambda=1$.

For *t* = 1, 2, …, *D*

(a) Randomly sample an integer *k* from the Poisson distribution: $Poisson(\lambda)$.

(b) Update the weak classifier h_t using the sample (*x*, *y*) *k* times

(c) If the weak classifier h_t correctly classifies the sample (*x*, *y*), i.e. $h_t(x)=sign(y)$, then

the weighted correct classification counter λ_t^{sc} is updated by $\lambda_t^{sc}\leftarrow\lambda_t^{sc}+\lambda$;

the approximate weighted false classification rate ε_t is updated by:

$$\varepsilon_t = \frac{\lambda_t^{sw}}{\lambda_t^{sc} + \lambda_t^{sw}} \tag{16}$$

the weight of the sample (x, y) is updated by:

$$\lambda \leftarrow \lambda \left(\frac{1}{2(1-\varepsilon_t)} \right) \tag{17}$$

else

λ_t^{sw} is updated by $\lambda_t^{sw} \leftarrow \lambda_t^{sw} + \lambda$;

ε_t is updated using (16);

the weight of the sample (x, y) is updated by:

$$\lambda \leftarrow \lambda \left(\frac{1}{2\varepsilon_t} \right). \tag{18}$$

(d) The weight α_t for the weak classifier h_t is:

$$\alpha_t = \log\left(\frac{1-\varepsilon_t}{\varepsilon_t} \right). \tag{19}$$

The weight α_t reflects the importance of feature t for detecting intrusions.

Step 3: The final strong classifier is defined by:

$$H(x) = sign\left(\sum_{t=1}^{D} \alpha_t h_t(x) \right) = sign\left(\sum_{t=1}^{D} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t} \right) h_t(x) \right). \tag{20}$$

The differences between the offline Adaboost algorithm and the online Adaboost algorithm are as follows:

- In the offline Adaboost algorithm, the weak classifiers are constructed in one step using all the training samples. In the online Adaboost algorithm, the weak classifiers are updated gradually as the training samples are input one by one.
- In the offline Adaboost algorithm, all the sample weights are updated simultaneously according to the classification results of the optimal weak classifier for the samples. In the above online Adaboost algorithm, the weight of the current sample changes as the weak classifiers are updated one by one using the current sample.
- In the offline Adaboost algorithm, the weighted classification error for each weak classifier is calculated according to its classification results for the whole set of the training samples. In online Adaboost, the weighted classification error for each weak classifier is estimated using the weak classifier's classification results only for the samples which have already been input.
- The number of weak classifiers is not fixed in the offline Adaboost algorithm. In the online Adaboost, the number of weak classifiers is fixed, and equal to the dimension of the feature vectors.
- When only a few training samples have been input, the online ensemble classifier is less accurate than

the offline ensemble classifier. As the number of training samples increases, the accuracy of the online ensemble classifier gradually increases until it approximates to the accuracy of the offline ensemble classifier [43].

The limitation of the traditional online Adaboost algorithm is that for each new input sample, all the weak classifiers are updated in a predefined order. The accuracy of the ensemble classifier depends on this order. There is a tendency to over-fit to the weak classifiers which are updated first.

3.2.2. A new online Adaboost algorithm

To deal with the limitation of the traditional online Adaboost, our new online Adaboost algorithm selects a number of weak classifiers according to a certain rule and updates them simultaneously for each input training sample. Let S^+ and S^- be, respectively, the numbers of the normal and attack samples which have already been input. Let Ω be the number of the samples, each of which is correctly classified by the previous strong classifier which has been trained before the sample is input. Let λ_t^{sc} be the sum of the weights of the input samples that are correctly classified by the weak classifier h_t. Let λ_t^{sw} be the sum of the weights of the input samples that are mistakenly classified by h_t. Let C_t be the number of the input samples which are correctly classified by h_t. All the parameters S^+, S^-, Ω, λ_t^{sc}, λ_t^{sw}, and C_t are initialized to 0. For a new sample (x, y), $y \in \{1, -1, -2, \cdots\}$, the strong classifier is updated online by the following steps:

Step 1: Update the parameter S^+ or S^- by:

$$\begin{cases} S^+ \leftarrow S^+ + 1 & if\ y = 1 \\ S^- \leftarrow S^- + 1 & else \end{cases}. \tag{21}$$

Initialize the weight λ of the new sample by:

$$\begin{cases} \lambda = (S^+ + S^-)/S^+ & if\ \ y = 1 \\ \lambda = (S^+ + S^-)/S^- & else \end{cases} \tag{22}$$

where λ follows the change of S^+ and S^-, in favor of balancing the proportion of the normal samples and the attack samples to ensure that the sum of the weights of the attack samples equals to the sum of the weights of the normal samples.

Step 2: Calculate the combined classification rate υ_t for each weak classifier h_t by:

$$\upsilon_t = (1-\rho)\varepsilon_t - \rho sign(y)h_t(x) = (1-\rho)\frac{\lambda_t^{sw}}{\lambda_t^{sc} + \lambda_t^{sw}} - \rho sign(y)h_t(x) \tag{23}$$

where ρ is a weight ranging in $(0, 0.5]$. The rate υ_t combines the "historical" false classification rate ε_t of h_t for the samples input previously and the result of h_t for the current sample (*x*, *y*). The rate υ_t is more effective than ε_t, as it gives h_t whose ε_t is high more chance to be updated and then increases the detection rate of h_t.

Define $\min \upsilon$ by:

$$\min \upsilon = \min_{t \in \{1,2,\ldots D\}} \upsilon_t \tag{24}$$

The weak classifiers $\{h_t\}_{t=1,\ldots,D}$ are ranked in the ascending order of υ_t and then the weak classifiers are represented by $\{h_{r_1}, h_{r_2}, \cdots, h_{r_D}\}$, $r_i \in \{1, 2, \cdots, D\}$.

Step 3: The weak classifies whose combined classification rates υ_{ri} are not larger than 0.5 are selected from $\{h_{r_1}, h_{r_2}, \cdots, h_{r_D}\}$. Each of these selected weak classifiers h_{ri} is updated using (*x*, *y*) in the following way:

Compute the number P_i of iterations for h_{ri} by:

$$P_i = Integer(P \exp(-\gamma(\upsilon_i - \min \upsilon))) \tag{25}$$

where γ is an attenuation coefficient and *P* is a predefined maximum number of iterations.

Repeat the following steps (a) and (b) P_i times in order to update h_{ri}:

(a) Online update h_{ri} using (*x*, *y*).

(b) Update the sample weight λ, λ_t^{sc}, and λ_t^{sw}:

If $sign(y) = h_{ri}(x)$, then

$$\begin{cases} C_t \leftarrow C_t + 1 \\ \lambda_{ri}^{sc} \leftarrow \lambda_{ri}^{sc} + \lambda \\ \lambda \leftarrow \lambda \left(\dfrac{1 - 2\rho}{2(1 - \upsilon_{ri})} \right) \end{cases}. \tag{26}$$

(λ is decreased)

If $sign(y) \neq h_{ri}$, then

$$\begin{cases} \lambda_{ri}^{sw} \leftarrow \lambda_{ri}^{sw} + \lambda \\ \lambda \leftarrow \lambda \left(\dfrac{1 + 2\rho}{2\upsilon_{ri}} \right) \end{cases}. \tag{27}$$

(λ is increased)

Step 4: Update C_t, λ_t^{sc} and λ_t^{sw} for each h_t of the weak classifiers whose combined classification rates are larger than 0.5 ($\upsilon_t > 0.5$):

if $sign(y) = h_t(x)$, then

$$\begin{cases} C_t \leftarrow C_t + 1 \\ \lambda_t^{sc} \leftarrow \lambda_t^{sc} + \lambda \end{cases} \tag{28}$$

else $\lambda_t^{sw} \leftarrow \lambda_t^{sw} + \lambda$.

Step 5: Update the parameter Ω: If the current sample is correctly classified by the previous strong classifier which has been trained before the current sample (*x*, *y*) is input, then $\Omega \leftarrow \Omega + 1$.

Step 6: Construct the strong ensemble classifier:

Calculate the ensemble weight α_t^* of h_t by

$$\alpha_t^* = \beta \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) + (1-\beta)\log\left(\frac{C_t}{\Omega}\right). \tag{29}$$

The α_t^* is normalized to α_t by:

$$\alpha_t = \frac{\alpha_t^*}{\sum_{i=1}^{D} \alpha_i^*}. \tag{30}$$

The strong classifier $H(x)$ is defined by:

$$H(x) = sign(\sum_{t=1}^{D} \alpha_t h_t(x)). \tag{31}$$

We explain two points for the above online Adaboost algorithm:

- The attenuation coefficient γ controls the number of weak classifiers which are chosen for further updating. When γ is very small, P_i is large and all the chosen weak classifiers are updated using the new sample. When γ is very large, P_i equals 0 if υ_t is large. As a result, for very large γ, only the weak classifiers with the small υ_t are updated.
- The ensemble weight α_t, which is calculated from ε_t and $\log(C_t/\Omega)$, is different from the one in the traditional Adaboost algorithm. The term $\log(C_t/\Omega)$ which is called a "contributory factor" represents the contribution rate of h_t to the strong classifier. It can be used to tune the ensemble weights to attain better detection performance.

3.3. Adaptable initial sample weights

We use the detection rate and the false alarm rate to evaluate the performance of the algorithm for detecting network intrusions. It is necessary to pay more attention to the false alarm rate, because in real applications most network behaviors are normal. A high false alarm rate wastes resources, as each alarm has to be checked. For Adaboost-based learning algorithms, the detection rate and the false alarm rate depend on the initial weights of the training samples. So we propose to adjust the initial sample weights in order to balance the detection rate and the false alarm rate. We introduce a parameter $r \in (0,1)$ for setting the initial weight λ of each training sample:

$$\lambda = \begin{cases} \frac{N_{normal} + N_{intrusion}}{N_{normal}} \cdot r & for\ normal\ connections \\ \frac{N_{normal} + N_{intrusion}}{N_{intrusion}} \cdot (1-r) & for\ network\ intrusions \end{cases}. \tag{32}$$

where N_{normal} and $N_{intrusion}$ are approximated using the numbers of normal samples and attack samples which have been input online to train the classifier. The sums of the weights for the normal samples and the attack samples are $(N_{normal}+N_{intrusion})\cdot r$ and $(N_{normal}+N_{intrusion})\cdot(1-r)$ respectively. Through adjusting the value of the parameter *r*, we change the importance of normal samples or attack samples in the training process, and then make a tradeoff between the detection rate and the false alarm rate of the final detector. The selection of *r* depends on the proportion of the normal samples in the training data, and the requirements for the detection rate and the false alarm rate in specific applications.

3.4. Local parameterized models

Subsequent to the construction of the weak classifiers and the online Adaboost learning, a local parameterized detection model φ is formed in each node. The local model consists of the parameters φ_w of the weak classifiers and the parameters φ_d for constructing the Adaboost strong classifier: $\varphi=\{\varphi_w,\varphi_d\}$. The parameters for each decision stump-based weak classifier include the subsets C_i^f and C_n^f for each categorical feature and the thresholds *v* for each continuous feature. The parameters for each GMM-based weak classifier include a set of GMM parameters $\varphi_w=\{\theta_j^c \mid j=1,2,...,D;\ \ c=1,-1,-2,...\}$. The parameters of the strong classifier for the online Adaboost algorithm include a set of ensemble weights $\varphi_d=\{\alpha_t \mid t=1,2,...,D\}$ for the weak classifiers.

The parameters in the decision stump-based weak classifiers depend on the differences between normal behaviors and attacks in each component of the feature vectors. The parameters in the GMM-based weak classifiers depend on the distributions of the different types of attacks and normal behaviors in each component of the feature vectors. The parameters in the strong classifier depend on the significances of individual feature components for intrusion detection. The local detection models capture the distribution characteristics of observed mixed-attribute data in each node. They can be exchanged between the different nodes.

Compared with the non-parametric distributed outlier detection algorithm proposed by Otey *et al.* [4], where a large amount of statistics about frequent itemsets need to be shared among the nodes, the parametric models are not only concise to be suitable for information sharing, but also very useful to generate global intrusion detection models.

4. Global Detection Models

The local parametric detection models are shared among all the nodes, and combined in each node to produce a global intrusion detector using a small number of samples left in the node (most samples in the node are removed, in order to adapt to changing network environments). This global intrusion detector is more accurate than the local detectors which may be only adequate for specific attack types, due to the limited training data available at each node.

Kittler *et al.* [11] develop a common theoretical framework for combining classifiers using the product rule, the sum rule, the max rule, the min rule, the median rule, and the majority vote rule. It is shown that the sum rule

has a better performance than others. Some researchers fuse multi-classifiers by combine the output results of all the classifiers into a vector, and then using a classifier, such as SVM or ANN, to classify the vectors.

The combination of the local intrusion detection models has two problems. First, there may be large performance gaps between the local detection models for different types of attacks, especially for new attack types which have not appeared previously. So, the sum rule may not be the best choice for combining the local detectors. Second, some of the local models may be similar for a test sample. If the results of the local models for the test sample are combined into a vector, the dimension of the vector has to be reduced to choose the best combination of the results from local models.

To solve the above two problems, we combine the particle swarm optimization (PSO) and SVM algorithms, in each node, to construct the global detection model. The PSO [12, 13] is a population search algorithm which simulates the social behavior of birds flocking. The SVM is a learning algorithm based on the structural risk minimization principle from statistical learning theory. It has a good performance even if the set of the training samples is small. We use the PSO to search for the optimal local models and the SVM is trained using the samples left in a node. Then, the trained SVM is used as the global model in the node. By combining the searching ability of the PSO and the learning ability of the SVM for small sample sets, a global detection model can be constructed effectively in each node.

The state of a particle *i* used in the PSO for one of the *A* nodes is defined as $X_i = (x_1, x_2, \cdots, x_A)$, $x_j \in \{0,1\}$, where "$x_j = 1$" means that the *j*-th local model is chosen, and "$x_j = 0$" means the *j*-th local model is not chosen. For each particle, a SVM classifier is constructed. Let *L* be the number of local detection nodes chosen by the particle state X_i. For each network connection sample left in the node, a vector $(r_1, r_2, \cdots, r_L)$ is constructed, where r_j is the result of the *j*-th chosen local detection model for the sample (corresponding to (20) and (31)):

$$r_j = \sum_{t=1}^{D} \rho_t h_t(x) \tag{33}$$

where *D* is the dimension of network connection feature vectors. These results are in the range [-1, 1]. All the vectors corresponding to the small number of samples left in the node, together with the attributions (normal connections or attacks) of the samples, are used to train the SVM classifier.

Each particle *i* has a corresponding fitness value which is evaluated by a fitness model $f(X_i)$. Let $\gamma(X_i)$ be the detection rate of the trained SVM classifier for particle ***i***, where the detection rate is estimated using another set of samples in the node. The fitness value is evaluated as:

$$f(X_i) = \tau * \gamma(X_i) + (1-\tau)\log\frac{A-L}{A} \tag{34}$$

where τ is a weight ranging between 0 and 1.

For each particle *i*, its individual best state S_i^l which has the maximum fitness value is updated in the *n*-th PSO iteration using the following equation:

$$S_i^l = \begin{cases} X_{i,n} & \text{if} \quad f(X_{i,n}) > f(S_i^l) \\ S_i^l & else \end{cases} \tag{35}$$

The global best state S^g in all the particles is updated by:

$$S^g = \arg\max_{S_i^l} f(S_i^l) \tag{36}$$

Each particle in the PSO is associated with a velocity which is modified according to its current best state and the current best state among all the particles:

$$V_{i,n+1} = F\left(wV_{i,n} + c_1\mu_1(S_i^l - X_{i,n}) + c_2\mu_2(S^g - X_{i,n})\right) \tag{37}$$

where w is the inertia weight which is negatively correlated with the number of iterations; c_1 and c_2 are acceleration constants called the learning rates; μ_1 and μ_2 are relatively independent random values in the range [0,1]; and $F()$ is a function which confines the velocity within a reasonable range:

$$F(V) = \begin{cases} V & if \ \|V\| \le V_{\max} \\ V_{\max} & else \end{cases}. \tag{38}$$

The states of the particles are evolved using the updated velocity :

$$X_{i,n+1} = X_{i,n} + V_{i,n+1}. \tag{39}$$

In theory, PSO can find the global optimum after sufficiently many iterations. In practice, a local optimum may be obtained, due to an insufficient number of iterations. PSO has a higher probability of reaching the global optimum than gradient descent approaches or particle filtering approaches.

The SVM and PSO-based fusion algorithm is outlined as follows:

Step 1: Initialization: The particles $\left\{X_{i,0}\right\}_{i=1}^{\mathbb{Q}}$ are randomly chosen in the particle space, where $\mathbb{Q}$ is the number of particles. $S_i^l = X_{i,0}$. A SVM classifier is constructed for each particle, and the detection rate $\gamma(X_{i,0})$ of the SVM classifier is calculated. The fitness value $f(X_{i,0})$ is calculated using (34). The global best state S_g is estimated using (36). $n = 0$.

Step 2: The velocities $\left\{V_{i,n+1}\right\}_{i=1}^{\mathbb{Q}}$ are updated using (37).

Step 3: The particles' states $\left\{X_{i,n+1}\right\}_{i=1}^{\mathbb{Q}}$ are evolved using (39).

Step 4: The SVM classifier is constructed for each particle, and the detection rate $\gamma(X_{i,n+1})$ is calculated. The fitness values $f(X_{i,n+1})$ are calculated using (34). The individual best states $\left\{S_i^l\right\}_{i=1}^{\mathbb{Q}}$ are update using (35). The global best sate S^g is updated using (36).

Step 5: $n \leftarrow n+1$. If $f(S^g) > max\_fintness$ or the predefined number of iterations is achieved, then the particle evolution process terminates and the SVM classifier corresponding to S^g is chosen

as the final classifier — the global model in the node; otherwise go to Step 2 for another loop of evolution.

When certain conditions are met, nodes may transmit their local models to each other. Then, each node can construct a customized global model using a small set of training samples randomly selected from the "historical" training samples in the node according to the proportion of various kinds of the network behaviors. Once local nodes gain their own global models, the global models are used to detect intrusions: for a new network connection, the vector of the results from the local models chosen by the global best particle is used as the input to the global model whose result determines whether the current network connection is an attack.

We explain two points:

- A uniform global model can be constructed for all the local nodes. But, besides the parameters of local models, small sets of samples should be shared through communications between the nodes.
- The computational complexity of the PSO for searching for the optimal local models determines the scale of network for our global model. The computational complexity of the PSO is $O(\mathbb{Q}IA^2\Gamma^2)$ where I is the number of iterations, and Γ is the number of the training samples.

5. Experiments

We use the KDD (Knowledge Discovery and Data Mining) CUP 1999 dataset [14, 15, 16, 45] to test our algorithms. This dataset is still the most trustful and credible public benchmark dataset [53, 54, 55, 56, 57, 58, 59] for evaluating network intrusion detection algorithms. In the dataset, 41 features including 9 categorical features and 32 continuous features are extracted for each network connection. Attacks in the dataset fall into four main categories:

- DOS: denial-of-service;
- R2L: unauthorized access from a remote machine, e.g. guessing password;
- U2R: unauthorized access to local super-user (root) privileges;
- Probe: surveillance and other probing, e.g. port scanning.

Each of the four categories contains some low-grade attack types. The test dataset includes some attack types that do not exist in the training dataset. The numbers of normal connections and each category of attacks in the training and test datasets are listed in Table 1.

Table 1. The KDD CUP 1999 data set

Categories	Training data	Test data
Normal	97278	60593
DOS	391458	223298
R2L	1126	5993
U2R	52	39
Probing	4107	2377
Others	0	18729
Total	494021	311029

In the following, we first introduce the performances of our online learning-based intrusion detection

algorithms: one with decision stumps and the traditional online Adaboost process, and the other with online GMMs and our proposed online Adaboost process. Then, the performance of our PSO and SVM-based distributed intrusion detection algorithm is evaluated.

5.1. Local models

For the online local models, the ability to handle mixed features, the effects of parameters on detection performance, and the ability to learn new types of attacks are estimated in succession. Finally, the performances of our algorithms are compared with the published performances of the existing algorithms.

1) Handling mixed features

Table 2 shows, respectively, the results of the classifier with decision stumps and the traditional online Adaboost and the classifier with online GMMs and our online Adaboost, when only continuous features or both continuous features and categorical features are used respectively, tested on the KDD training and test data sets respectively. It is seen that the results obtained by using both continuous and categorical features are much more accurate than the results obtained by only using continuous features. This shows the ability of our algorithms to handle mixed features in network connection data.

Table 2. The results obtained by using only continuous features or both continuous and categorical features

Algorithms	Features	Training data		Test data	
		Detection rate	False alarm rate	Detection rate	False alarm rate
Decision stump + Traditional online Adaboost	Only Continuous	98.68%	8.35%	90.05%	13.76%
	Continuous + Categorical	98.93%	2.37%	91.27%	8.38%
Online GMM + Our online Adaboost	Only Continuous	98.79%	7.83%	91.33%	11.34%
	Continuous + Categorical	99.02%	2.22%	92.66%	2.67%

2) Effects of parameters on detection performance

In the following, we illustrate the effects of important parameters, i.e. the adaptable initial weight parameter r in (32), the data distribution weight coefficient "1/M" in (4), the parameter ρ in (23), the parameter β in (29), and the attenuation coefficient parameter γ in (25).

Table 3. The results of the algorithm with decision stumps and the traditional online Adaboost when adaptable initial weights are used

r	Training data		Test data	
	Detection rate	False alarm rate	Detection rate	False alarm rate
0.25	98.71%	2.38%	90.83%	8.42%
0.3	98.69%	0.78%	90.69%	2.29%
0.35	98.63%	0.77%	89.80%	1.92%
0.4	98.53%	0.73%	89.66%	1.88%
0.5	98.16%	0.16%	88.97%	0.37%

Table 3 shows the results of the algorithm with decision stumps and the traditional online Adaboost when the adaptable initial weights formulated by (32) are used. When r is increased the detection rate and the false alarm rate are decreased. An appropriate setting of r is 0.3, as with this value of r, the false alarm rate of 2.28% is much smaller than the false alarm rate of 8.42% obtained by choosing r as 0.25, while an acceptable detection rate of 90.69% is still obtained. As shown in Table 2, when an identical initial weight is used for each training sample in the traditional online Adaboost algorithm using decision stumps as weak classifiers, a high detection

rate of 91.27% is obtained on the test data, but the false alarm rate is 8.38%, which is too high for intrusion detection. After adaptable initial weights are introduced, although the detection rate is slightly decreased, the false alarm rate is much decreased, producing more suitable results for intrusion detection. This illustrates that adjustable initial weights can be used effectively to balance the detection rate and the false alarm rate.

Table 4 shows the results of the algorithm with online GMMs and our online Adaboost when the data distribution weight "1/*M*" in (4) and adaptable initial weights are used or not used. It is seen that when the parameter *M* is not used, the false alarm rate is 54.15%, which is too high for intrusion detection. When the parameter *M* is used, the false alarm rate falls to 4.82%. When adaptable initial weights are used (*r* is set to 0.35) the false alarm rate further falls to 1.02%, while an acceptable detection rate is maintained.

Table 4. The results of our online Adaboost algorithm based on online GMMs with or without adaptable initial weights and the data distribution weight "1/*M*"

Parameters	Training data		Test data	
	Detection rate	False alarm rate	Detection rate	False alarm rate
Without the data distribution weight "1/*M*"	99.86%	48.85%	97.49%	54.15%
With "1/*M*" and without adaptable initial weights	99.22%	8.10%	91.74%	4.82%
With "1/*M*" and adaptable initial weights	98.91%	2.65%	90.65%	1.02%

Parameters ρ in (23) and β in (29) influence the detection results. Table 5 shows the detection rates and false-alarm rates of the algorithm with online GMMs and our online Adaboost when $\rho = 0$ or 0.1 and $\beta = 1$ or 0.8. From this table, the following two points are deduced:

- When ρ is set to 0, the false alarm rate is very high. This is because the weak classifiers are chosen only according to the false classification rate ε_t, without taking into account the current sample. Adopting the measure υ_t of the combined classification rate balances the effects of "historical" samples and the current sample, and then results in more accurate results.
- When β is set to 1, the detection rate is too low. This is because under this case only ε_t is used to weight the weak classifiers, i.e. every weak classifier is weighted in the same way as in the offline Adaboost and the traditional online Adaboost. In the online training process, weighting every weak classifier only using ε_t causes over-fitting to some weak classifiers. Namely, the ensemble weights of weak classifiers based only on ε_t are not accurate. Weighting weak classifiers by combining ε_t and "contributory factors" yields the better performance.

Table 5. The results of the algorithm with online GMMs and our online Adaboost when $\rho = 0$ or 0.1 and $\beta = 0.8$ or 1

	Detection rate	False alarm rate
$\rho = 0$ $\beta = 0.8$	96.22%	77.66%
$\rho = 0.1$, $\beta = 1$	85.36%	0.60%
$\rho = 0.1, \beta = 0.8$	91.15%	1.69%

The attenuation coefficient γ in (25) is important for the performance of our online Adaboost algorithm. Table 6 shows the detection rates and the false alarm rates of our online Adaboost algorithm based on GMMs, when γ ranges from 10 to 50. When $\gamma \in \{20, 25, 30\}$, the updating times P_t is better chosen for the weak classifiers and thus good results are obtained. In fact, when γ is set to 25, the detection rate reaches 91.15% with a false-alarm rate of 1.69%. This is a preferable result compared with those obtained when γ is set to 10 or 50.

Table 6. The detection results of our online Adaboost algorithm based on online GMMs with varying γ

γ	Detection rate	False alarm rate
10	92.50%	12.87%
20	90.61%	1.17%
25	91.15%	1.69%
30	90.55%	1.26%
40	88.28%	0.37%
50	24.33%	0.34%

3) Learning new attack types

The ability to learn effectively from new types of attacks and then correctly detect these types of attacks is an attractive and important characteristic of our online Adaboost-based intrusion detection algorithms.

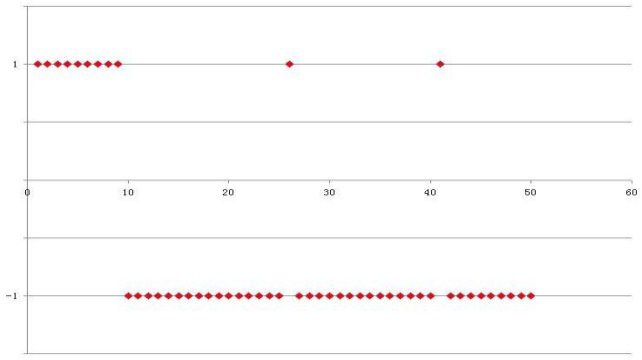


Fig. 2. Online learning with decision stumps and the traditional online Adaboost for “guess-passwd” attacks

Figs. 2 and 3 show the online learning process of the ensemble classifier with decision stumps and the traditional online Adaboost with respect to two attack types, “guess-passwd” and “httptunnel”, which have not appeared before in the training data. In the figures, the horizontal coordinate indicates the number of samples of “guess-passwd” or “httptunnel”, which have appeared in the learning process; and the vertical coordinate indicates the class label $\tilde{y}$ predicted by the detector for a sample before it takes part in the learning process, where “$\tilde{y} = 1$” means that the sample is mistakenly classified as a normal connection, and “$\tilde{y} = -1$” means that it is correctly classified as a network attack. It is seen that at the beginning of learning for new types of attacks, the classifier frequently classifies mistakenly samples of these new attack types as normal network connections. After some samples of these types of attacks take part in the online learning of the classifier, these new attack

types are much more correctly detected. These two examples show the effectiveness of online learning of the algorithm with decision stumps and the traditional online Adaboost for new attack types.

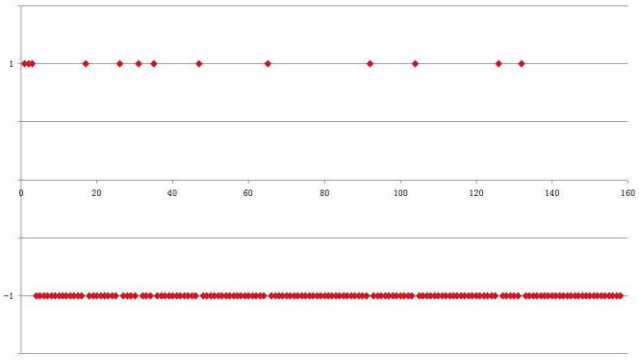


Fig. 3. Online learning with decision stumps and the traditional online Adaboost for “httptunnel” attacks

Figs. 4 and 5 show the online learning process of the ensemble classifier with online GMMs and our online Adaboost for the two attack types, “guess-passwd” and “httptunnel”, which have not appeared before in the training data. It is seen that after some samples of “guess-passwd” and “httptunnel” occur in the online learning, these two types of samples are then detected much more correctly by the ensemble classifier. Compared with the classifier with decision stumps and the traditional online Adaboost, the classifier with online GMMs and our online Adaboost learns the new types of attacks more quickly and accurately. This is because the classifier with online GMMs and our online Adaboost has more powerful learning ability based on modeling the distributions of each type of attacks.

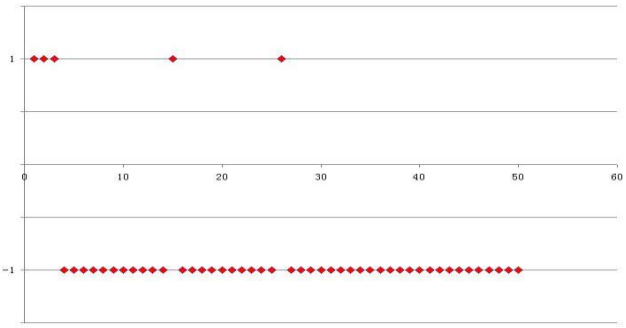


Fig. 4. Online learning with online GMMs and our online Adaboost for “guess-passwd” attacks

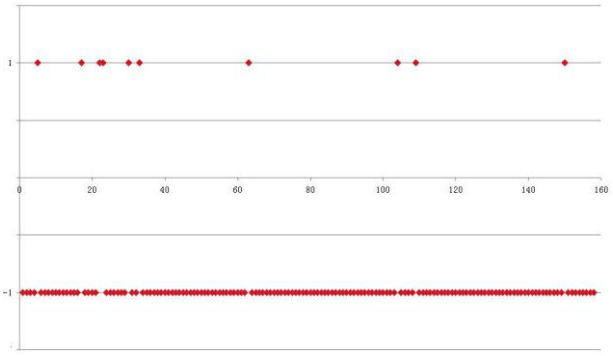


Fig. 5. Online learning with online GMMs and our online Adaboost for “httptunnel” attacks

4) Comparisons

We first compare the detection results obtained from part of the labeled KDD CUP 1999 dataset, then the results obtained from the whole labeled KDD CUP 1999 dataset.

Some intrusion detection algorithms are tested on a smaller dataset that contains only three common attack types, namely Neptune, Satan, and Portsweep, in order to examine the algorithm's ability to detect specific types of attack. The samples in the dataset are randomly selected from the KDD CUP 1999 data set. Table 7 lists the numbers of the training and test samples of all the types in this smaller dataset. Table 8 shows the detection results of the algorithm based on hybrid neural networks [21], the algorithm based on the hierarchical SOM [18], the algorithm with decision stumps and the traditional online Adaboost, and the algorithm with online GMMs and our online Adaboost, all tested on the smaller data set. It is seen that the detection rate of our algorithm with decision stumps and the traditional online Adaboost for the specific types of attacks is 99.55%, which is close to the highest detection rate for the two algorithms in [18] and [21]. The false alarm rate of this algorithm is only 0.17%, which is much less than the means in the false alarm rate ranges for the algorithms in [18] and [21]. Our algorithm with online GMMs and our online Adaboost even obtains more accurate results. It is shown that our online Adaboost-based algorithms are effective for these three types of attacks.

Table 7. A smaller data set

Categories	Training data	Test data
Normal	20676	60593
Neptune	22783	58001
Satan	316	1633
Portsweep	225	354
Total	44000	120581

Table 8. The detection results for some specific attacks

Algorithms	Detection rate	False alarm rate
Hybrid neural networks [21]	90%-99.72%	0.06%-4.5%
Hierarchical SOM [18]	99.17%-99.63%	0.34%-1.41%
Our algorithm (decision stumps + traditional online Adaboost)	99.55%	0.17%
Our algorithm (online GMM + our online Adaboost)	99.99%	0.31%

We also compare our online Adaboost-based algorithms with other recently published algorithms for intrusion detection. The algorithms are tested on the whole labeled KDD CUP 1999 dataset. Table 9 lists the detection results of the following algorithms: the clustering-based algorithm in [3], the *k*-**NN**-based algorithm in [20], the SVM-based approach in [20], the algorithm based on SOMs in [5], the genetic clustering-based algorithm in [17], the hierarchical SOM-based algorithm in [18], the bagged C5 algorithm in [19], the offline Adaboost-based algorithm in [39], the Mercer kernel ART algorithm in [8], our algorithm using decision stumps and the traditional online Adaboost, and our algorithm using online GMMs and our online Adaboost. The Mercer kernel ART algorithm in [8] and our algorithms are online learning algorithms and the others are offline learning algorithms. From the table, the following useful points are deduced:

- Overall, the detection accuracies of the online Adaboost-based algorithms are comparable with those of

the other algorithms.

- Compared with the offline algorithms, our algorithms not only gain satisfactory detection rates while keeping low false alarm rates, but also adaptively modify the local models in online mode. This adaptability is very important for intrusion detection applications.
- The detection accuracies of the online Adaboost-based algorithms are comparable with the detection accuracies of the Adaboost-based algorithm in batch mode.
- Our online Adaboost-based algorithms outperform the online algorithms in [8]. In particular, lower false alarm rates are obtained.
- The detection accuracy of the algorithm using decision stumps and the traditional online Adaboost is lower than the algorithm using online GMMs and our proposed online Adaboost. The reason for this is that the algorithm based on GMMs models online the distribution of each type of attacks and also updates the weak classifiers in a more effective way, which obtains more accurate weights for the weak classifiers.

Table 9. Comparison between algorithms tested on the KDD CUP 1999 data set

Algorithms		Detection rate	False alarm rate
Offline	Clustering [3]	93%	10%
	K-NN [20]	91%	8%
	SVM [20]	91%-98%	6%-10%
	SOM [5]	89%-90.6%	4.6%-7.6%
	Genetic clustering[17]	79.00%	0.30%
	Hierarchical SOM [18]	90.94%-93.46%	2.19%-3.99%
	Bagged C5 [19]	91.81%	0.55%
	Offline Adaboost [39]	90.04-90.88%	0.31-1.79%
Online	Mercer kernel ART [8]	90.00-94.00%	2.9-3.4%
	Our algorithm(decision stumps+traditional Adaboost)	90.13%	2.23%
	Our algorithm(Online GMMs+our Adaboost)	90.61-91.15%	1.17-1.69%

Moreover, Brugger and Chow [50] have assessed the performance of the Snort algorithm, a typical pattern matching-NIDS, on the DARPA 1998 dataset which is the dataset of the raw data of the KDD CUP 1999 dataset. It is shown that Snort has a very high alarm rate. The results of the Snort algorithm are less accurate than those of the machine learning algorithms listed in Table 9.

Our online Adaboost-based algorithms are implemented on a Pentium IV computer with 2.6GHZ CPU and 256M RAM, using MATLAB 7. The mean training time for our algorithm with decision stumps and the traditional online Adaboost is only 46s and that for our algorithm with online GMMs and our online Adaboost is 65 minutes, using all 494,021 training samples. This is an empirical demonstration that the computational complexity of our algorithm with decision stumps and the traditional online Adaboost is relatively low, due to very simple operations for the decision stumps. The computational complexity of our algorithm with online GMMs and our online Adaboost is moderate. In [46], the least training times required for the SOM and improved competitive learning neural network are, respectively, 1057s and 454s, only using 101,000 samples for training.

The bagged C5 algorithm [19, 47] took a bit more than a day on a machine with a two-processor ultra-sparc2 (2~300Mhz) and 512M main memories. The RSS-DSS algorithm [35] requires 15 minutes to finish the learning process on a 1 GHz Pentium III laptop with 256M RAM. In [4], 8 minutes are taken for processing the training data in an eight-node cluster, where each node has dual 1GHz Pentium III processors and 1GB memory, running Red Hat Linux7.2, while 212 minutes are taken for the algorithm in [48]. The mean training time for the offline Adaboost is 73s [39].

5.2. PSO and SVM-based global models

The proposed distributed intrusion detection algorithm is tested with 6 nodes. The KDD CUP 1999 training dataset is split into six parts and each part of data is used to construct a local detection model in a detection node. In this way, the size of training data in each node is small, with the result that accurate local intrusion detectors cannot be found in some nodes. In each node, a global model is constructed using the six local models.

To simulate a distributed intrusion detection environment, the four kinds of attacks: neptune, smurf, portsweep and satan in the KDD CUP 1999 training dataset are used for constructing local detection models, as samples of these four kinds take up 98.46% of all the samples in the KDD training dataset. Table 10 shows the training sets used for constructing the global models in the 6 nodes. It is seen that the sizes of the training sets are comparatively small. Node 1 has not the portsweep and satan attack samples; Node 2 has not the satan samples; and Node 3 has not the portsweep attack samples. Table 11 shows the test set used in all the nodes.

Table 10. The training sets in the 6 nodes

Attack types \ Nodes	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Neptune	5000	1000	1000	3500	2500	1500
smurf	5000	1000	1000	3500	2500	1500
portsweep	0	8000	0	1500	2500	3500
satan	0	0	8000	1500	2500	3500
Nomal	10000	10000	10000	10000	10000	10000

Table 11. The test set in each node

neptune	smurf	portsweep	satan	Normal
58001	164091	10413	15892	60593

In the experiments, the inertia weight w in (37) varies from 1.4 to 0.4, according to the following equation:

$$w = (w - 0.4)\frac{Titer - Iter}{Titer} + 0.4 \tag{40}$$

where $Titer$ is the predefined maximum number of iterations and $Iter$ is the current iteration number.

Tables 12 and 13 show, using our algorithm with decision stumps and the traditional online Adaboost and our algorithm with online GMMs and our online Adaboost respectively, the detection rates and false alarm rates of the local models and those of the PSO and SVM-based global models, as well as those obtained by combining the local models using the SVM algorithm, the sum rule, and the majority vote rule. From the tables, the following three points are deduced:

- It is seen that, overall, our PSO and SVM-based combination algorithm greatly increases the detection

rate and decreases the false alarm rate for each node after the local models with only a small number of parameters are shared.

- In node 2, the detection rate of the PSO-SVM method is slightly less than the detection rate of the local model, but the false alarm rate of the PSO-SVM method is much less than the false alarm rate of the local model. In node 4, the alarm rate of the PSO-SVM method is slightly more than the alarm rate of the local model, but the detection rate of the PSO-SVM method is higher than the detection rate of the local model. Thus, the PSO-SVM method is more accurate than the local models in nodes 2 and 4.
- Our PSO and SVM-based algorithm is superior to the SVM algorithm, the sum rule and the majority vote rule for combining local detection models. The performance disparities between different local models ensure that the sum rule and the majority vote rule are not suitable for distributed intrusion detection. The SVM may not find the optimal local model combination to improve the performance. Through dynamically combining a small portion of the local models to obtain the global model, our PSO and SVM-based algorithm effectively chooses the optimal local model combination, reduces the time consumption for detecting intrusions, and then achieves a better performance.

Table 12. The results for the 6 local detection nodes using our algorithm with decision stumps and the traditional online Adaboost

Methods	Performance	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Average
Local model	Detection rate	99.48%	99.96%	99.98%	97.01%	97.13%	99.96%	98.92%
	False alarm rate	24.07%	9.64%	18.45%	0.42%	0.49%	8.28%	10.23%
PSO-SVM	Detection rate	99.59%	99.70%	99.67%	99.78%	99.65%	99.74%	99.69%
	False alarm rate	0.38%	0.44%	0.43%	0.44%	0.41%	0.48%	0.43%
SVM	Detection rate	99.76%	99.78%	97.13%	99.78%	99.78%	97.13%	98.89%
	False alarm rate	0.44%	0.44%	0.42%	0.44%	0.45%	0.42%	0.44%
Sum rule	Detection rate	99.96%	99.96%	99.96%	99.96%	99.96%	99.96%	99.96%
	False alarm rate	1.98%	1.98%	1.98%	1.98%	1.98%	1.98%	1.98%
Majority vote	Detection rate	99.96%	99.96%	99.96%	99.96%	99.96%	99.96%	99.96%
	False alarm rate	7.95%	7.95%	7.95%	7.95%	7.95%	7.95%	7.95%

Table 13. The results for 6 local detection nodes using our algorithm with online GMMs and our online Adaboost

Methods	Performance	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Average
Local model	Detection rate	96.19%	33.66%	33.9%	99.72%	99.82%	33.77%	66.18%
	False alarm rate	0.35%	0.39%	0.56%	0.34%	0.37%	0.40%	0.40%
PSO-SVM	Detection rate	99.61%	99.64%	99.89%	99.84%	99.84%	99.83%	99.78%
	False alarm rate	0.3%	0.31%	0.39%	0.35%	0.34%	0.33%	0.34%
SVM	Detection rate	97.46%	99.76%	99.85%	99.83%	99.84%	99.85%	99.43%
	False alarm rate	0.31%	0.35%	0.36%	0.33%	0.34%	0.34%	0.34%
Sum	Detection rate	99.76%	99.76%	99.76%	99.76%	99.76%	99.76%	99.76%
	False alarm rate	0.37%	0.37%	0.37%	0.37%	0.37%	0.37%	0.37%
Majority vote	Detection rate	33.79%	33.79%	33.79%	33.79%	33.79%	33.79%	33.79%
	False alarm rate	0.34%	0.34%	0.34%	0.34%	0.34%	0.34%	0.34%

As stated in the last paragraph of Section 4, a uniform global model can be constructed for all the local nodes, if small sets of samples are shared among the nodes, sacrificing the privacy of raw data. We also test the uniform global model using the dataset of the four kinds of attacks: neptune, smurf, portsweep and satan in the

KDD CUP 1999 set. The training set used for constructing the global model only contains 4000 randomly chosen samples, and the test set for the global model contains 284672 samples of the four types of attacks and all the normal samples. Table 14 shows the attack types used to train local models in each node. It is seen that each node has only one or two types of attack. Tables 15 and 16 show, using our algorithm with decision stumps and the traditional online Adaboost and our algorithm with online GMMs and our online Adaboost respectively, the detection rates and false alarm rates of the local models in each node and those of the PSO and SVM-based global model, as well as those obtained by combining local models using the sum rule and the SVM algorithm. It is seen that our PSO and SVM-based algorithm greatly increases the detection accuracy in each node and our PSO and SVM-based algorithm obtains more accurate results than the sum rule and the SVM algorithm.

Table 14. The attack types used to train the local models in each node

Local models	Types of attack
Node 1	Neptune
Node 2	Smurf
Node 3	Portsweep
Node 4	Satan
Node 5	neptune, smurf
Node 6	portsweep, satan

Table 15. The results of the uniform global model based on our algorithm with decision stumps and the traditional online Adaboost when some samples are shared among nodes

Detection models		Detection rate	False alarm rate
Local models	Node 1	26.47%	0.12%
	Node 2	73.20%	0.22%
	Node 3	2.19%	0.1%
	Node 4	8.77%	0.01%
	Node 5	99.62%	0.32%
	Node 6	5.32%	0.11%
Global model (PSO-SVM)		99.99%	1.35%
Sum rule		9.10%	0.01%
SVM		98.98%	1.41%

Table 16. The results of the uniform global model based on our algorithm with online GMMs and our online Adaboost when some samples are shared among nodes

Detection models		Detection rate	False alarm rate
Local models	Node 1	26.48%	0.08%
	Node 2	70.16%	0.01%
	Node 3	7.92%	0.17%
	Node 4	0.81%	0.01%
	Node 5	99.62%	0.21%
	Node 6	26.77%	1.80%
Global model (PSO-SVM)		99.99%	0.37%
Sum rule		26.37%	0.01%
SVM		99.99%	0.39%

The distributed outlier detection algorithm in [4] has a detection rate of 95% and a false alarm rate of 0.35% for the KDD CUP 1999 training data set, however, the algorithm in [4] requires a data preprocessing procedure in which the attack data are packed into bursts in the data set. An intrusion is marked as detected if at least one instance in a burst is flagged as an outlier. Furthermore, the high communication cost and the need to share a large number of the original network data confine the application of the algorithm in [4] to network intrusion detection. In our proposed algorithm, no assumptions are made about bursts of attacks in the training data and no private data are shared among the nodes, and the only communication is the concise local model parameters.

6. Conclusion

In this paper, we have proposed online Adaboost-based intrusion detection algorithms, in which decision stumps and online GMMs have been used as weak classifiers for the traditional online Adaboost and our proposed online Adaboost respectively. The results of the algorithm using decision stumps and the traditional online Adaboost have been compared with the results of the algorithm using online GMMs and our online Adaboost. We have further proposed a distributed intrusion detection framework, in which the parameters in the online Adaboost algorithm have formed the local detection model for each node, and local models have been combined into a global detection model in each node using a PSO and SVM-based algorithm. The advantages of our work are as follows: 1) Our online Adaboost-based algorithms successfully overcome the difficulties in handling the mixed-attributes of network connection data. 2) The online mode in our algorithms ensures the adaptability of our algorithms to the changing environments: the information in new samples is incorporated online into the classifier, while maintaining high detection accuracy. 3) Our local parameterized detection models are suitable for information sharing: only a very small number of data are shared among nodes. 4) No original network data are shared in the framework, so that the data privacy is protected. 5) Each global detection model improves considerably on the intrusion detection accuracy for each node.

References

[1] D. Denning, "An intrusion detection model," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 2, pp. 222-232, Feb. 1987.

[2] J.B.D. Caberera, B. Ravichandran, and R.K. Mehra, "Statistical traffic modeling for network intrusion detection," *in Proc. of Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 466-473, 2000

[3] W. Lee, S.J. Stolfo, and K. Mork, "A data mining framework for building intrusion detection models," *in Proc. of IEEE Symposium on Security Privacy*, pp. 120-132, May 1999.

[4] M.E. Otey, A. Ghoting, and S. Parthasarathy, "Fast distributed outlier detection in mixed-attribute data sets," *Data Ming and Knowledge Discovery*, vol. 12, no. 2-3, pp. 203-228, May 2006.

[5] H.G. Kayacik, A.N. Zincir-heywood, and M.T. Heywood, "On the capability of an SOM based intrusion detection system," *in Proc. of International Joint Conference on Neural Networks*, vol. 3, pp. 1808-1813, July 2003.

[6] P.Z. Hu and M.I. Heywood, "Predicting intrusions with local linear model," *in Proc. of International Joint Conference on Neural Networks*, vol. 3, pp. 1780-1785, July 2003.

[7] Z. Zhang and H. Shen, "Online training of SVMs for real-time intrusion detection," *in Proc. of Advanced Information Networking and Applications*, vol. 2, pp. 568-573, 2004.

[8] H. Lee, Y. Chung, and D. Park, "An adaptive intrusion detection algorithm based on clustering and kernel-method," *in Proc .of International Conference on Advanced information Networking and Application*, pp.603-610, 2004.

[9] W. Lee and S.J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information an System Security*, vol. 3, no. 4, pp. 227-261, Nov. 2000.

[10] A. Fern and R. Givan, "Online ensemble learning: an empirical study," *in Proc. of International Conference on Machine Learning*, pp. 279-286, 2000.

[11] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. "On combining classifiers," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, no.3, pp. 226-238, March 1998.

[12] J. Kennedy, "Particle swarm optimization," *in Proc. of IEEE International Conference on Neural Networks*, Perth, pp. 1942-1948, 1995.

[13] Y. Shi and R.C. Eberhart, "A modified particle swarm optimizer," *in Proc. of IEEE International Conference on Evolutionary Computation*, Anchorage, USA, pp.69-73, 1998.

[14] S. Stofo et al. "The third international knowledge discovery and data mining tools competition," The University of California. 2002. Available: http://kdd.ics.uci.edu/databases/kddCup99/kddCup99.h-tml.

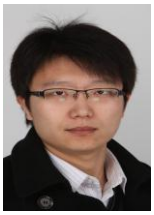
[15] S. Mukkamala, A.H. Sung, and A. Abraham. "Intrusion detection using an ensemble of intelligent paradigms," *Network and Computer Applications*, vol. 28, no. 2, pp. 167-182, April 2005.
[16] R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *ACM Trans. on Information and System Security*, vol. 34, no. 4, pp. 579-595, Oct. 2000.
[17] Y.G. Liu, K.F. Chen, X.F. Liao, and W. Zhang. "A genetic clustering method for intrusion detection," *Pattern Recognition*, vol. 37, no. 5, pp. 927-942, May 2004.
[18] S.T. Sarasamma, Q.A. Zhu, and J. Huff. "Hierarchical Kohonenen net for anomaly detection in network security," *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 35, no. 2, pp. 302-312, April 2005.
[19] B. Pfahringer. "Winning the KDD99 classification cup: bagged boosting," *SIGKDD Explorations*, vol. 1, no. 2, pp. 65-66, 2000.
[20] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data," *in Applications of Data Mining in Computer Security*, Chapter 4. D. Barbara and S. Jajodia (editors) Kluwer, 2002
[21] C. Jirapummin, N. Wattanapongsakorn, and P. Kanthamanon. "Hybrid neural networks for intrusion detection systems," *in Proc. of International Technical Conference on Circuits/Systems, Computers and Communications*, pp. 928-931, July 2002.
[22] J. Li and C. Manikopoulos, "Novel statistical network model: the hyperbolic distribution," *IEE Proceeding on Communications*, vol. 151, no. 6, pp. 539-548, Dec. 2004.
[23] T. Peng, C. Leckie, and K. Ramamohanarao, "Information sharing for distributed intrusion detection systems," *Journal of Network and Computer Applications*, vol. 30, no. 3, pp. 877-899, Aug. 2007.
[24] M. Qin and K. Hwang, "Frequent episode rules for internet anomaly detection," *in Proc. of IEEE International Symposium on Network Computing and Applications*, pp. 161-168, 2004.
[25] S. Zanero and S.M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," *in Proc. of ACM Symposium on Applied Computing*, pp. 412-419, 2004.
[26] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, and M. Embrechts, "Network-based intrusion detection using neural networks," *in Proc. of Artificial Neural Networks in Engineering*, vol. 12, New York, pp. 579-584, Nov. 2002.
[27] J.J.M. Bonifacio, A.M. Cansian, A.C.P.L.F. De Carvalho, and E.S. Moreira, "Neural networks applied in intrusion detection systems," *in Proc. of IEEE International Joint Conference on Neural Networks*, vol. 1, pp. 205-210, 1998.
[28] C. Zhang, J. Jiang, and M. Kamel, "Intrusion detection using hierarchical neural networks," *Pattern Recognition Letters*, vol. 26, no. 6, pp. 779-791, 2005.
[29] S.J. Han and S.B. Cho, "Evolutionary neural networks for anomaly detection based on the behavior of a program," *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, vol. 36, no. 3, pp. 559-570, June 2006.
[30] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," *in Proc. of International Joint Conference on Neural Networks*, vol. 2, pp. 1702-1707, 2002.
[31] J. Mill and A. Inoue, "Support vector classifiers and network intrusion detection," *in Proc. of International Conference on Fuzzy Systems*, vol. 1, pp. 407-410, 2004.
[32] J. Xian, F. Lang, and X. Tang, "A novel intrusion detection method based on clonal selection clustering algorithm," *in Proc. of International Conference on Machine Learning and Cybernetics*, vol. 6, pp. 3905-3910, 2005.
[33] S. Jiang, X. Song, H. Wang, J. Han, and Q. Li, "A clustering-based method for unsupervised intrusion detections," *Pattern Recognition Letters*, vol. 27, no. 7, pp. 802-810, May 2006.
[34] A.J. Hoglund, K. Hatonen, and A.S. Sorvari, "A computer host-based user anomaly detection system using the self-organizing map," *in Proc. of International Joint Conference on Neural Networks*, vol. 5, pp. 411-416, 2000.
[35] D. Song, M.I. Heywood, and A.N. Zincir-Heywood, "Training genetic programming on half a million patterns: an example from anomaly detection," *IEEE Trans. on Evolutionary Computation*, vol. 9, no. 3, pp. 225-239, June 2005.
[36] S. Parthasarathy, A. Ghoting, and M.E. Otey, "A survey of distributed mining of data streams," Data streams: models and algorithms, C. C. Aggarwal, Eds. New York: Springer-Verlag, Nov. 2006.
[37] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119-139, August 1997.
[38] N. Oza. "Online Ensemble Learning," PhD thesis, University of California, Berkeley, 2001.
[39] W.M. Hu, W. Hu, and S. Maybank, "Adaboost-based algorithm for network intrusion detection," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 2, pp. 577-583, April 2008.
[40] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 5, pp. 649-659, Sep. 2008
[41] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 39, no. 1, pp. 1-38, 1977.
[42] H. Grabner and H. Bischof, "On-line boosting and vision," *in Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 260-267, 2006.
[43] N. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," *in Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, pp. 359-364, 2001.
[44] Y. Lei, X.Q. Ding, and S.J. Wang, "Visual tracker using sequential Bayesian learning: discriminative, generative and hybrid," *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 38, no. 6, pp. 1578-1591, Dec. 2008.
[45] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory," *ACM Trans. on Information and System Security*, vol. 3, no. 4, pp. 262-294, Nov. 2000.
[46] J.Z. Lei and A. Chorbani, "Network intrusion detection using an improved competitive learning neural network," *in Proc. of Second Annual Conference on Communication Networks and Services Research*, vol. 4, pp. 190-197, May 2004.
[47] C. Elkan. "Results of the KDD99 classifier learning contest," *SIGKDD Explorations*, vol. 1, no. 2, pp. 63-64, 2000.
[48] S. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," *in Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 29-38, 2003.
[49] S. Mabu, C. Chen, N. Lu, K. Shimada, and K. Hirasawa, "An intrusion-detection model based on fuzzy class-association-rule

mining using genetic network programming," *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 1, pp. 130-139, Jan. 2011.
[50] S.T. Brugger and J. Chow, "An assessment of the DARPA IDS evaluation dataset using Snort," Technical Report CSE-2007-1, University of California, Jan. 2007.
[51] S. Panigrahi and S. Sural, "Detection of database intrusion using a two-stage fuzzy system," *Information Security*, Lecture Notes in Computer Science, vol. 5735, pp. 107-120, 2009.
[52] D. Smallwood and A. Vance, "Intrusion analysis with deep packet inspection: increasing efficiency of packet based investigations," *in Proc. of International Conference on Cloud and Service Computing*, pp. 342-347, Dec. 2011.
[53] S. Mabu, C. Chen, N. Lu, K. Shimada, and K. Hirasawa, "An intrusion-detection model based on fuzzy class-association-rule mining using genetic network programming," *IEEE Trans. on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, vol. 41, no. 1, pp. 130-139, Jan. 2011.
[54] C. Otte and C. Stormann, "Improving the accuracy of network intrusion detectors by input-dependent stacking," *Integrated Computer-Aided Engineering*, vol. 18, pp. 291-297, 2011.
[55] K.-C. Khor, C.-Y. Ting, S. Phon-Amnuaisuk, "A cascaded classifier approach for improving detection rates on rare attack categories in network intrusion detection," *Applied Intelligence*, vol. 36, pp. 320-329, 2012.
[56] B. Zhang, "A heuristic genetic neural network for intrusion detection," *in Proc. of International Conference on Internet Computing and Information Services*, pp. 510-513, Sept. 2011.
[57] C.-F. Tsai, J.-H. Tsai, and J.-S. Chou, "Centroid-based nearest neighbor feature representation for e-government intrusion detection," *in Proc. of World Telecommunications Congress*, pp. 1-6, March 2012.
[58] J. Gao, W. Hu, X. Zhang, and X. Li, "Adaptive distributed intrusion detection using parametric model," *in Proc. of IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, vol. 1, pp. 675-678, Sept. 2009.
[59] P. Prasenna, A.V.T. RaghavRamana, R. KrishnaKumar, and A. Devanbu, "Network programming and mining classifier for intrusion detection using probability classification," *in Proc. of the International Conference on Pattern Recognition, Informatics and Medical Engineering*, pp. 204-209, March 2012.
[60] K. Rieck, "Machine learning for application-layer intrusion detection," Dissertation, Fraunhofer Institute FIRST & Berlin Institute of Technology, Berlin Germany, 2009.

Acknowledgments

This work is partly supported by NSFC (Grant No. 60935002), the National 863 High-Tech R&D Program of China (Grant No. 2012AA012504), the Natural Science Foundation of Beijing (Grant No. 4121003), and The Project Supported by Guangdong Natural Science Foundation (Grant No. S2012020011081).

Weiming Hu received the Ph.D. degree from the department of computer science and engineering, Zhejiang University in 1998. From April 1998 to March 2000, he was a postdoctoral research fellow with the Institute of Computer Science and Technology, Peking University. Now he is a professor in the Institute of Automation, Chinese Academy of Sciences. His research interests are in visual surveillance, and filtering of Internet objectionable information.



Jun Gao has a B.S. in Automation Engineering from BeiHang University, Beijing, China, and a Ph.D. in Computer Science from Institute of Automation, Chinese Academy of Sciences. His major research interests include machine learning, data mining, and network information security.

Yanguo Wang received the BSc degree in information and computing science from Sun Yat-sen University, Guangzhou, China, in 2005. In 2008, he received the MSc degree at the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences. He is currently a project manager in the Institute of Infrastructure Inspection, China Academy of Railway Sciences, Beijing. His research interests include pattern recognition, computer vision, and data mining.

Ou Wu received the BS degree in Electrical Engineering from Xi'an Jiaotong University in 2003. He received the MS degree and PhD degree both from pattern recognition and machine intelligence from National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences in 2006 and 2011, respectively. Now, he is an Assistant Professor in NLPR. His research interests include information filtering, data mining, and web vision computing.

Stephen Maybank received a BA in Mathematics from King's college Cambridge in 1976 and a PhD in computer science from Birkbeck college, University of London in 1988. Now he is a professor in the School of Computer Science and Information Systems, Birkbeck College. His research interests include the geometry of multiple images, camera calibration, visual surveillance etc.