Expert Systems with Applications 38 (2011) 3492–3498

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers

Ming-Yang Su *

Department of Computer Science and Information Engineering, Ming Chuan University, 5 Teh Ming Road, Gwei Shan District, Taoyuan 333, Taiwan

A R T I C L E I N F O

Keywords:
KNN (k-nearest-neighbor) classification
Genetic algorithm
NIDS (network intrusion detection system)
Network security
DoS attacks
Feature selection
Feature weighting

A B S T R A C T

This study proposed a method which can detect large-scale attacks, such as DoS attacks, in real-time by weighted KNN classifiers. The key factor for designing an anomaly-based NIDS is to select significant features for making decisions. Not only is excellent detection performance required, but real-time processing is also demanded for most NIDSs. A good feature selection policy, which can choose significant and as few as possible features, plays a key role for any successful NIDS. The study proposed a genetic algorithm combined with KNN (k-nearest-neighbor) for feature selection and weighting. All initial 35 features in the training phase were weighted, and the top ones were selected to implement NIDSs for testing. Many DoS attacks were applied to evaluate the systems. For known attacks, an overall accuracy rate as high as 97.42% was obtained, while only the top 19 features were considered. For unknown attacks, an overall accuracy rate of 78% was obtained using the top 28 features.

1. Introduction

E-commerce systems are based upon Internet use, which provides open and easy communications on a global basis. Since the Internet is unregulated, unmanaged and uncontrolled, it introduces a wide range of risks and threats to the systems operating on it. This is the reason that the network intrusion detection systems (NIDSs) have been emerging recently. NIDSs are traditionally divided into two broad categories: misuse detection and anomaly detection. Misuse detection aims to detect known attacks by characterizing the rules that govern these attacks. Thus, rules update is particularly important, and consequently, new definitions are frequently released by NIDS vendors. However the rapid emergence of new vulnerabilities makes misuse detection difficult to trust. Anomaly detection is designed to capture any deviation from the profiles of normal behavior patterns. Anomaly detection is much more suitable than misuse detection to detect unknown or novel attacks, but it has the potential to generate too many false alarms. Therefore, this study proposed a system for anomaly detection.

Most NIDSs emphasize on effectiveness but neglect efficiency, especially for anomaly-based NIDSs. Usually, effectiveness is measured by detection rate, false alarm rate, etc, and efficiency is measured by response time while an attack occurs. Since too many features for an anomaly-based NIDS would not necessarily guarantee good performance, it certainly delays the detection engine's ability to make a decision. Thus, how to select fewer but significant features becomes vital. Furthermore, features should be weighted because their contributions to correct classification are different from each other. That is the goal of the paper. Since DoS/DDoS (Denial-of-Service/Distributed DoS) attacks are prevalent and becoming one of the main threads to E-commerce systems, this system was evaluated by DoS/DDoS attacks.

For an anomaly-based NIDS, the most difficult part is to present the normal profile, which depends on the policy of feature weighting and selection. In past studies, many good anomaly-based NIDSs have focused on system architectures or detection engine designs (e.g. Carl, Kesidis, Brooks, & Rai, 2006; Gavrilis & Dermatas, 2005; Kulkarni & Bush, 2006; Wang, Zhang, & Shin, 2004); only few of them have focused on the feature weighting and selection, such as Mukkamala and Sung (2002), Sung and Mukkamala (2003), Lee, Chung, and Shin (2006), Abbes, Bouhoula, and Rusinowitch (2004), Stein, Chen, Wu, and Hua (2005), Hofman, Horeis, and Sick (2004), Middlemiss and Dick (2003), Liao and Rao Vemuri (2002). Most of them have applied KDD CUP99 dataset for experiments. In order to promote the comparison of different works in IDS (intrusion detection system) area, the Lincoln Laboratory at MIT, under the Defense Advanced Research Project Agency (DARPA) and Air Force Research Laboratory sponsorship, constructed and distributed the first standard dataset for evaluation of computer network IDS (DARPA, XXXX) in 1998. In 1999, the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining with the purpose of demonstrating the learning contest,

* Tel.: +886 3 3507001; fax: +886 3 3593874.
E-mail address: minysu@mail.mcu.edu.tw

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.08.137

collected and generated TCP dump data provided by the aforementioned DARPA (XXXX) in the form of train-and-test sets. The above dataset is named as KDD CUP99 dataset (KDD CUP, 1999a).

Mukkamala and Sung (2002) applied the technique of SVMs (Support Vector Machines) to rank the 41 features provided by KDD CUP99 dataset (KDD CUP, 1999b). Mukkamala and Sung ranked the features again by both SVMs and neural networks in Sung and Mukkamala (2003). Lee et al. (2006) discussed the feature selection based on genetic algorithm combined with relief tree and genetic algorithm combined with Naïve Bayesian network. They also used KDD CUP99 dataset for experiments. Abbes et al. (2004) and Stein et al. (2005) applied decision trees to design their detection engines. Tree nodes were selected by genetic algorithm in Abbes et al. (2004), and information gain mixed with gain ratio and Gini index in Stein et al. (2005). A self-created dataset was experimented with in Abbes et al. (2004), while KDD CUP99 dataset was used in Stein et al. (2005). Hofman et al. (2004) applied genetic algorithm combined with an RBF (radial basis function) network to feature selection, and took seven attacks out of KDD CUP99 dataset for an experiment. Finally, Middlemiss and Dick (2003) and Liao and Rao Vemuri (2002) both proposed a genetic algorithm combined with KNN for feature selection. The KDD CUP99 TCPDUMP was experimented with in Middlemiss and Dick (2003), while 1988 DARPA BSM audit data (DARPA, XXXX) was experimented with in Liao and Rao Vemuri (2002); here, BSM represents audit logs generated on the Sun machine using Solaris Basic Security Module (BSM). However, in Middlemiss and Dick (2003) the details about genetic algorithm and KNN were not mentioned, and in Liao and Rao Vemuri (2002) the authors regarded the BSM audit data as documents and applied document classification terms: TF&IDF (term frequency and inverse document frequency) to weight features.

Most of above researches evaluated their approaches by the KDD CUP99 dataset. This means that their researches were designed for off-line detection and thus can't meet real-time demands for NIDSs. This is because the announced 41 features in KDD CUP99 were derived from connections, not packets. In fact, the 41 features presented in KDD CUP99 are complicated and varied (KDD CUP, 1999b; Middlemiss & Dick, 2003). The first 9 of 41 are intrinsic features which describe the basic features of individual TCP connections and can be obtained from raw TCPDUMP files; features 10–22 are content-based features obtained by examining the data portion of a connection and suggested by domain knowledge; features 23–31 are traffic-based features that are computed using a two-second time window ("time-based"), while features 32–41 are also traffic-based features but computed using a window of 100 connections ("host-based"). Moreover, the collection of attacks appearing in KDD CUP99 is out of date; for instance, in total only 12 DoS attacks appear in the KDD CUP99 dataset.

All features used in this paper are derived from packet headers and gathered using a two-second time window. The method proposed in this paper can be implemented to be real-time, i.e. making a decision every two seconds. If necessary, the time window can be reduced to one second or half a second. Basically, the result of KNN classification was adopted to design the fitness function in a genetic algorithm, to evolve the weight vectors of features. When the evolution was terminating, an optimal weight vector was obtained. Then, the least weighted features were dropped one by one. As one feature was being dropped, the proposed system was retrained and re-evaluated. Finally, the optimal weight vector with the best number of features and their weights were obtained.

The remainder of this paper is organized as follows: Section 2 briefly introduces genetic algorithm and KNN; Section 3 describes the proposed system; Section 4 discusses the experimental results; and Section 5 provides the conclusion.

2. Background

This section will briefly introduce the genetic algorithm and KNN, since the approach adopted in this paper is a GA/KNN hybrid.

2.1. Genetic algorithm (GA)

GA is essentially a type of search algorithm used to solve a wide variety of problems. Its goal is to create optimal solutions to problems (Holland, 1992). A potential solution is encoded as a sequence of bits, characters or numbers. This unit of encoding is called a gene, and the encoding sequence is known as a chromosome.

GA begins with a set of chromosomes, called a population, and an evaluation function which measures the fitness value of each chromosome. Usually, an initial population of chromosomes is created by complete randomization. During evolution, chromosomes are evaluated by the fitness function. Based on their fitness values, better chromosomes are selected as parents by selection procedure, and then the parents perform crossover and mutation to form new children chromosomes. Finally, some chromosomes in the current generation are replaced by the new ones, if necessary, to form the next generation. The evolution continues until some predefined situation is met, such as the number of iterations reached or an acceptable fitness value appearing. The iteration loop of a basic genetic algorithm is illustrated in Fig. 1.

The design of the fitness function is the most important part in GA because a good one can significantly improve the outcome of the GA. This study applied the classification result of KNN to design the fitness function.

2.2. KNN (k-nearest-neighbor)

One common classification scheme based on the use of distance measures is that of the k-nearest-neighbor. The KNN technique assumes that the entire sampling set includes not only the data in the set, but also the desired classification for each item. When a classification is to be made for a new item, its distance to each item in the sampling set must be computed. Only the k closest entries in the sampling set are considered further. The new item is then classified to the class that contains the most items from this set of k

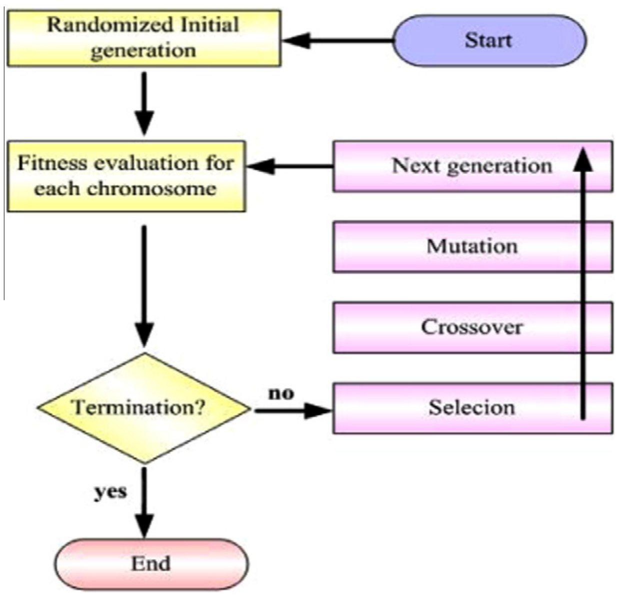
Randomized Initial generation
Start
Fitness evaluation for each chromosome
Next generation
Mutation
Crossover
Termination?
no
Selecion
yes
End

Fig. 1. Flowchart of GA.

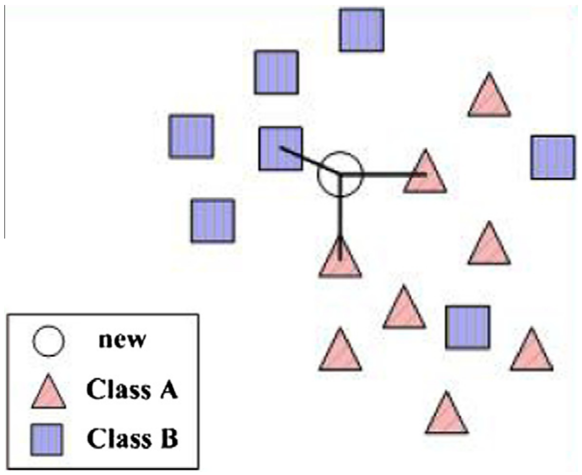

Fig. 2. An illustration of KNN for $k = 3$.

closest items. An example of KNN for $k = 3$ is illustrated in Fig. 2. In the example, the new instance is classified to class A.

The distance between two instances represents their similarity; hence, ingredients of an instance denote features. Euclidean distance is usually adopted in the KNN. For any two n-feature instances, say $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, their Euclidean distance is computed as:

$$dist(X,Y) = \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2 + \cdots + (x_n-y_n)^2} = \sqrt{1\cdot(x_1-y_1)^2 + 1\cdot(x_2-y_2)^2 + \cdots + 1\cdot(x_n-y_n)^2} \quad (1)$$

A major drawback of such distance measure, i.e., similarity measure, is that it regards all features equally. This can lead to poor similarity measure and thus cause misclassification. Such a phenomenon is especially severe when only a small subset of the features is really contributed to classification. Unfortunately, it is a reasonable assumption that for many kinds of classifications, it is uncertain which feature is more important than others or what features are absolutely useless.

3. The proposed approach

According to the framework of GA, the proposed NIDS is described by three parts in the section. The first subsection will present all features that are considered in the study; the second subsection will state the encoding of a chromosome and the fitness function; the third subsection will provide details on the selection, crossover and mutation in the GA.

3.1. Features for the proposed NIDS design

This study proposed a fast mechanism to detect DoS/DDoS attack from network traffic, so that all of the features come from the headers, including IP, TCP, UDP, ICMP, ARP, and IGMP. Table 1 lists all of the 35 features initially considered in the paper. Some of them are single condition, e.g., "S.IP slots hit", and some of them are compound ones, e.g., "S.IP + ACK Flag + ACK number". The former example denotes how many slots of IP addresses are hit by network packets according to their source IP addresses. A 32-bit IP address is mapped to one of 256 slots by its twice folded IP address, i.e., mapping IP:x1.x2.x3.x4 to slot (x1 $\oplus$ x2) $\oplus$ (x3 $\oplus$ x4). The later example denotes the maximal number of packets with source IP addresses belonging to the same slot and (ACK flag = 0 and ACK number $\neq$ empty).

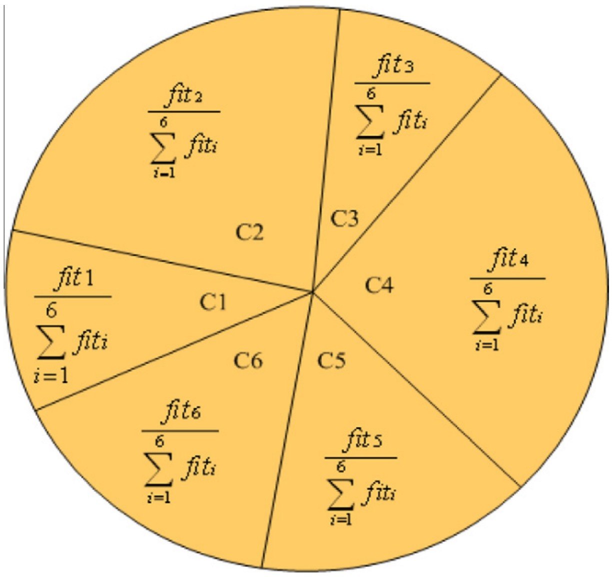
$\frac{fit_1}{\sum_{i=1}^{6} fit_i}$
$\frac{fit_2}{\sum_{i=1}^{6} fit_i}$
$\frac{fit_3}{\sum_{i=1}^{6} fit_i}$
$\frac{fit_4}{\sum_{i=1}^{6} fit_i}$
$\frac{fit_5}{\sum_{i=1}^{6} fit_i}$
$\frac{fit_6}{\sum_{i=1}^{6} fit_i}$
C1
C2
C3
C4
C5
C6

Fig. 3. Probabilities of chromosomes for parent in a 6-chromosome population.

Table 1
List of all features.

No.	Protocol	Feature	No.	Protocol	Feature
1	IP	S.IP slots hit	19	TCP	URG_Flag + URG_data count
2	TCP	S.IP + S.port slots hit	20	TCP	ACK_Flag + ACK count
3	TCP	S.IP + D.port slots hit	21	TCP	Checksum_error count
4	TCP	S.IP + SYN count	22	TCP	Same_length_interval count
5	TCP	S.IP + URG_Flag + URG_data count	23	TCP	Port (20) + length(>1400) count
6	TCP	S.IP + ACK_Flag + ACK count	24	UDP	S.port count
7	ARP	S.IP + ARP count	25	UDP	D.port count
8	IP	D.IP slots hit	26	UDP	Checksum_error count
9	IP	Header length != 20 count	27	UDP	Same_length_interval count
10	IP	MF_Flag count	28	ICMP	Type error count
11	IP	(Total length > 1400 || < 40) &&TTL == 64 count	29	ICMP	Checksum_error count
12	IP	Checksum_error count	30	ICMP	Length > 1000 count
13	TCP	S.port slots hit	31	IGMP	Type error count
14	TCP	D.port slots hit	32	IGMP	Checksum_error count
15	TCP	S.port count	33	IGMP	Length > 1000 count
16	TCP	D.port count	34	ARP	Size error count
17	TCP	Sequence_number == 0 count	35		Total packet number
18	TCP	SYN count			

Every experimental instance contains the 35 feature values observed from one time unit (2 s in this study) of network traffic. Since some feature values may be extremely large and dominate the distance calculation in Eq. (1), each feature value is normalized to the interval [0, 1] as follows:

$$\text{Normalization of } f_i = \frac{1 - e^{-kf_i}}{1 + e^{-kf_i}} \tag{2}$$

In Eq. (2), f_i is the observed value of feature i, e represents the Euler number ($e \approx 2.7182818284$), and k is a constant. Each feature has its own k value. The k constant is dependent on the average feature value of all normal instances in the sampling dataset. More precisely, k is heuristically set to let the normalized average value fall in the interval between 0.4 and 0.5.

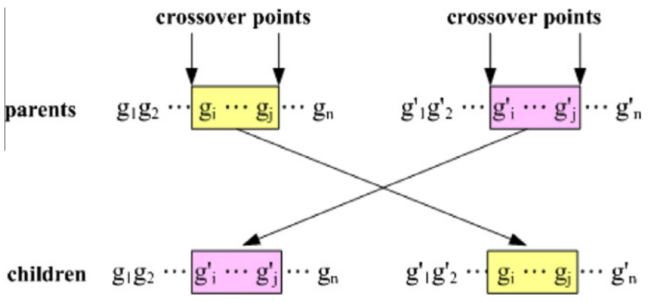
crossover points
crossover points
parents
g₁g₂ ··· gᵢ ··· gⱼ ··· gₙ
g'₁g'₂ ··· g'ᵢ ··· g'ⱼ ··· g'ₙ
children
g₁g₂ ··· g'ᵢ ··· g'ⱼ ··· gₙ
g'₁g'₂ ··· gᵢ ··· gⱼ ··· g'ₙ

Fig. 4. An illustration of two-point crossover.

Eq. (2) makes one instance independent to other instances during the normalization, so that the proposed method can be easily implemented to a real-time network intrusion detection system.

3.2. Genes, chromosomes and fitness function

Table 1 lists all of the 35 features considered in the paper. However, their weights should be different from each other. In Eq. (1), a major drawback is to treat every feature without difference. Some features may be more important than others in the KNN classification, or some of them can even be ignored. The goal of GA is to find an optimal weight vector, say $W = [w_1, w_2, \ldots, w_n]$, in which w_i represents the weight of feature i, $1 \leqslant i \leqslant n$. The W will influence the distance computation and finally promote the KNN classification. In this paper, for any two n-feature, e.g. $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, their distance is computed by the following equation.

$$dist(X, Y) = \sqrt{w_1(x_1 - y_1)^2 + w_2(x_2 - y_2)^2 + \cdots + w_n(x_n - y_n)^2} \tag{3}$$

Each w_i, a real number in the interval [0, 1], is a gene, and a feasible weight vector is a chromosome. Chromosomes in the initial population are randomly generated. After the evolution of GA in the training phase, an optimal weight vector can be obtained, which leads to the best result of KNN classification.

The fitness function applied to evaluate each chromosome during the evolution is shown in Eq. (4), which denotes the overall accuracy rate of KNN classification on the training dataset. In the equation, *Total* represents the total number of instances in the

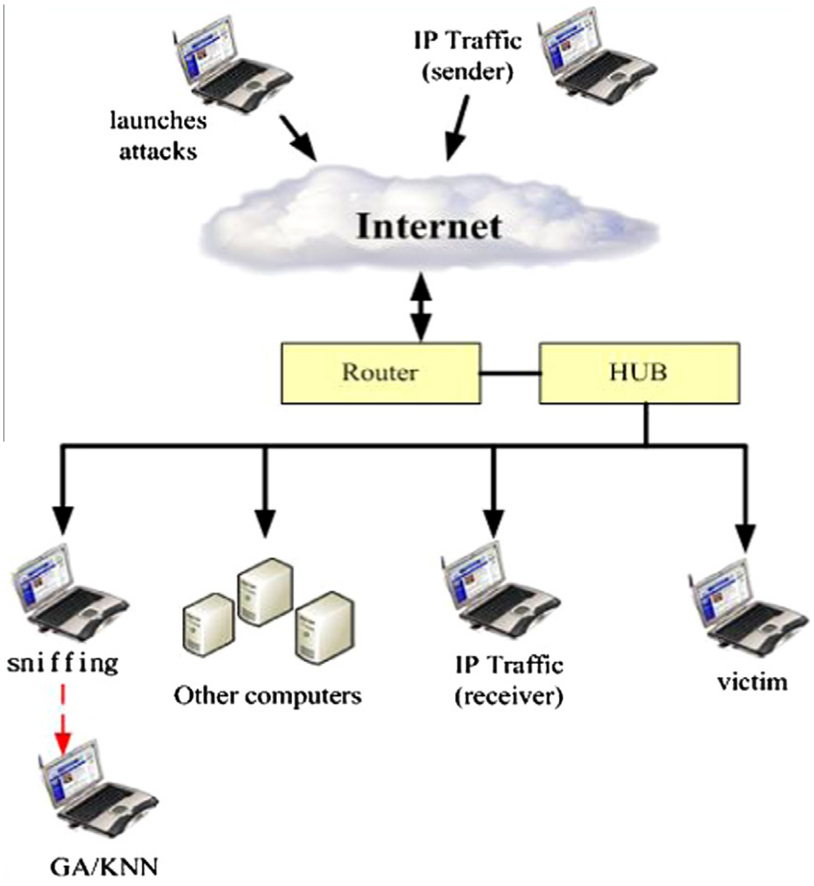

Fig. 5. Network topology for experiments.

labeled training dataset, and *FP* and *FN* represent the numbers of false positive instances and false negative instances, respectively.

$$\text{fitness} = \frac{Total - FP - FN}{Total} \tag{4}$$

Given a specific class, *C*, and a instance, x_i, that instance may or may not be assigned to that class while its actual membership may or may not be in that class. This gives us the four quadrants shown below.

True positive (TP): x_i predicted to be in *C* and is actually in it.
False positive (FP): x_i predicted to be in *C* and is not actually in it.
True negative (TN): x_i not predicted to be in *C* and is not actually in it.
False negative (FN): x_i not predicted to be in *C* and is actually in it.

Suppose that there are *s* instances for prediction. Then *s* is equal to the sum of instance number of TP, FP, TN, and FN. So, the numerator of Eq. (4) is equal to the sum of instance numbers of TP and TN. The fitness by Eq. (4) is named as *overall accuracy*.

3.3. Selection, crossover, and mutation

Roulette Wheel selection is used to choose two chromosomes from the parent population to produce children chromosomes. By Roulette Wheel, the larger fitness value a chromosome has, the more chance it gets to be chosen as parent. The example of the probabilities for chromosomes in a 6-chromosome population is illustrated in Fig. 3. In the figure, fit_i represents the fitness value of the *i*th chromosome.

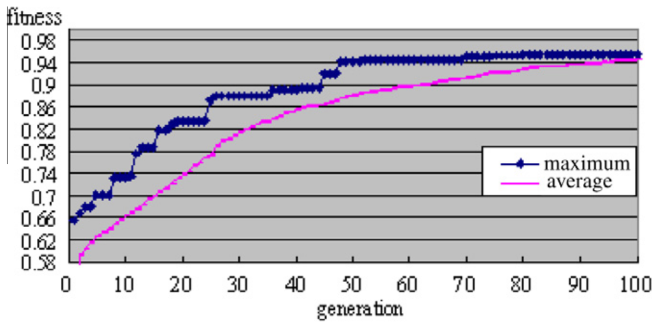

Fig. 6. Fitness vs. generation.

Two-point crossover was applied to exchange genes between parent chromosomes. An example is shown in Fig. 4. The genes between two crossover points of parents are exchanged to produce children. In a mutation, every chromosome was selected to have a 10% probability to mutate. For a chromosome that mutates, each gene is set to have a 10% probability to be randomly changed. If a mutated chromosome's fitness is larger than that of its original form, it then replaces the original one. Otherwise, this time of mutation is ignored.

4. Experimental results and analyses

The network topology for the experiment is shown in Fig. 5. A commercial application named IP Traffic (XXXX) was applied to produce background traffic, which can generate any amount of TCP/UDP/ICMP packets. Two hosts running IP Traffic played sender and receiver, respectively, and the receiver is deployed in the LAN and the sender transmitting packets through the Internet. By IP Traffic, users can choose protocols and set the contents of packets generated by mathematical laws (Pareto, Uniform, and Exponential) or derived from files, or just generated by a packet generator with configurable contents. Inter-packet delay and packet size could also be selected. In this experiment, the amount of network traffic remained from 0 to 80 Mbps through random connection and random flow size. One laptop in the Internet launched DoS attacks against the victim machine in the LAN. No firewall was applied, and all packets passed through the router. This system was coded by Microsoft Visual C++, and operated on a laptop with Windows XP.

All 65 attack programs applied in the experiments were downloaded from the VX Heavens website (http://vx.netlux.org/), which is maintained by the well-known Kaspersky. In the website, there are 207 programs for Win32 belonging to the DoS category. However, perhaps due to missing some specific DLLs, only 65 can be executed. All attack programs' names and experimental datasets can be found in the website at http://120.125.80.29/ga or http://120.125.80.30/ga.

Three kinds of datasets were used: sampling dataset, training dataset, and testing dataset. Every instance in all datasets is a 35-feature vector obtained from sniffing network packets for one time unit of 2 s. Certainly, each instance was normalized by Eq. (2). The sampling dataset contained 100 normal instances and 100 attack instances, which was used for KNN classification. The training dataset contained 600 normal instances and 600 attack instances, used to calculate the fitness value of chromosome in GA

Table 2
List of sorted features by their weights.

Rank	Protocol	Feature	Weight	Rank	Protocol	Feature	Weight
1	UDP	Same_length_interval count	0.9812	19	TCP	S.IP + SYN count	0.3091
2	IP	MF_Flag count	0.9682	20	UDP	Checksum_error count	0.2404
3	IGMP	Checksum_error count	0.9327	21	ICMP	Type error count	0.1592
4	TCP	S.IP + URG_Flag + URG_data count	0.8914	22	TCP	ACK_Flag + ACK count	0.1518
5	ARP	S.IP + ARP count	0.8767	23	UDP	S.port count	0.1203
6	IGMP	Length > 1000 count	0.8225	24	TCP	S.IP + ACK_Flag + ACK count	0.1056
7	TCP	Sequence_number == 0 count	0.7894	25	ICMP	Checksum_error count	0.0646
8	IGMP	Type error count	0.7586	26	TCP	Same_length_interval count	0.064
9	IP	(Total length > 1400|| < 40)&&TTL == 64 count	0.7047	27	UDP	D.port count	0.0544
10	ICMP	Length > 1000 count	0.6866	28	TCP	S.IP + D.port slots hit	0.0537
11	TCP	Checksum_error count	0.6561	29	TCP	D.port slots hit	0.0489
12	IP	Checksum_error count	0.6193	30	TCP	S.IP + S.port slots hit	0.0453
13	ARP	Size error count	0.5816	31	TCP	S.port slots hit	0.0317
14	TCP	URG_Flag + URG_data count	0.5274	32	TCP	S.port count	0.026
15	IP	D.IP slots hit	0.4591	33	TCP	D.port count	0.0217
16	TCP	Port (20) + length(>1400) count	0.4444	34	TCP	SYN count	0.0189
17	IP	S.IP slots hit	0.437	35		Total packet number	0.0128
18	IP	Header length != 20 count	0.3627				

Table 3
Performances for TestA and TestB as all 35 features being considered.

	TestA (%)	TestB (%)
TP	93.83	51.17
FP	0.33	1
TN	99.67	99
FN	6.17	48.83
Overall accuracy	96.75	75.08

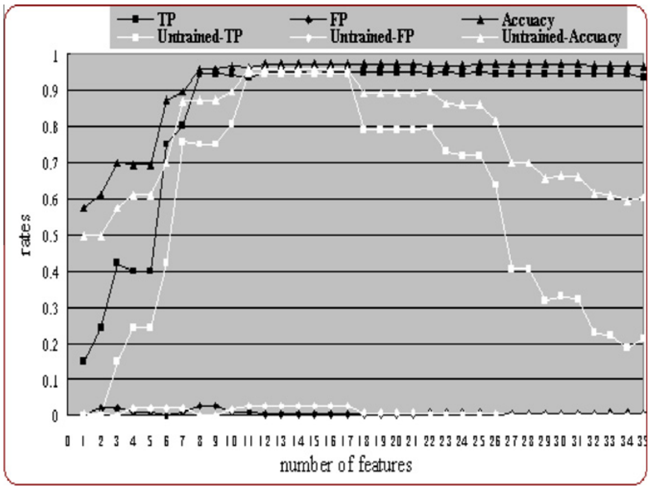

Fig. 7. Performances with different number of features for TestA.

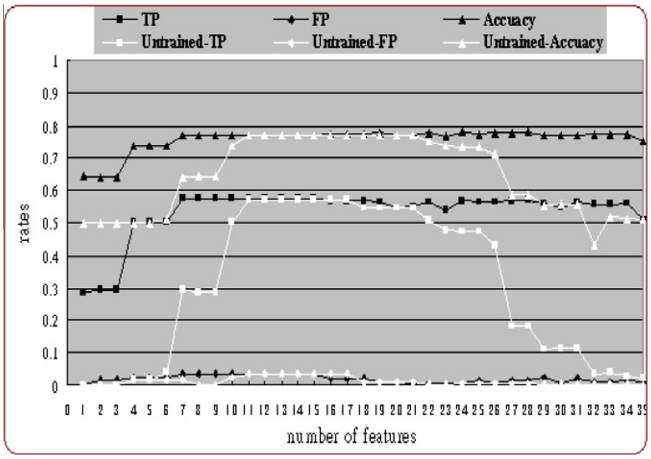

Fig. 8. Performances with different number of features for TestB.

evolution. All attack instances in the sampling and training datasets were derived from 20 out of the total 65 DoS programs. Using the same 20 attack programs to generate TestA dataset as known attacks, the other 45 programs were applied to generate a TestB dataset as unknown attacks. Both TestA and TestB contained 600 attack instances and 600 normal instances.

Initially, 30 chromosomes were randomly generated for the initial population. Using the training set, each chromosome, $W = [w_1, w_2, \ldots, w_{35}]$, was evaluated by computing its fitness value by Eq. (4), while the distance calculation followed Eq. (3). In order to obtain one chromosome's fitness value, Eq. (3) should be computed $1200 \times 200 = 240{,}000$ times because there were a total of 1200 instances in the training set and 200 instances in the sampling set. Fig. 6 shows the fitness values during the evolution for the case of 30 chromosomes and 100 generations. For each generation, only the best chromosome's fitness was considered. The experiment was repeated five times. In the figure, the bold line depicts the maximal fitness value among all runs, and the thin line depicts the average of all runs.

In Fig. 6, the maximal fitness value among all runs is 0.965, which was obtained by the chromosome given below. For each number, the digits before the colon denotes the feature number in Table 1 and the value after the colon denotes its corresponding weight.

[**1:**0.437, **2:**0.0453, **3:**0.0537, **4:**0.3091, **5:**0.8914, **6:**0.1056, **7:**0.8767, **8:**0.4591, **9:**0.3627, **10:**0.9682, **11:**0.7047, **12:**0.6193, **13:**0.0317, **14:**0.0489, **15:**0.026, **16:**0.0217, **17:**0.7894, **18:**0.0189, **19:**0.5274, **20:**0.1518, **21:**0.6561, **22:**0.064, **23:**0.4444, **24:**0.1203, **25:**0.0544, **26:**0.2404, **27:**0.9812, **28:**0.1592, **29:**0.0646, **30:**0.6866, **31:**0.7586, **32:**0.9327, **33:**0.8225, **34:**0.5816, **35:**0.0128].

Table 2 lists the sorted 35 features by their weights. While all of the 35 features were considered, performances for TestA and TestB are shown in Table 3. The true positive rate and false positive rate for TestA were 93.83% and 0.33%, respectively, and for that of TestB are 51.17% and 1%, respectively. The overall accuracy for TestA was 96.75% and for TestB was 75.08%. Almost all of false negatives for TestA were caused by the DoS.Win32.Pinger which performs like a normal ping executive, no mal-packet or violation of protocol, except for hundreds of ping packets per second.

Next, the features were moved one by one from the least one. As one feature was removed, we retrained (one run, 30 chromosomes, and 100 generations) for the best chromosome and re-evaluated by TestA and TestB. The results are shown in Fig. 7 for TestA and Fig. 8 for TestB.

For easy comparison, the performances of untrained weight vectors in both figures are presented by white lines. An untrained weight vector has all members to be 1, i.e., all features being treated equally. In Fig. 7, the best performance in terms of overall accuracy occurred as 19 features were applied, and it was 97.42%. Compared to Table 3, in which all 35 features were applied, the

Table 4
Feature numbers vs. performances for TestA.

#f.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
TP%	14.8	24.3	42.3	40.0	0.04	75.0	80.5	94.7	94.7	94.3	93.8	95.0	95.0	95.0	94.8	95.0	95.0	95.0
FP%	0.0	2.0	2.0	0.83	0.83	0.0	1.0	2.67	2.67	0.83	0.83	0.33	0.33	0.33	0.33	0.33	0.33	0.33
TN%	100	98.0	98.0	99.2	99.2	100	99.0	97.3	97.3	99.2	99.2	99.7	99.7	99.7	99.7	99.7	99.7	99.7
FN%	85.2	75.7	57.7	60.0	60.0	25.0	19.5	5.33	5.33	5.67	6.17	5.0	5.0	.05	5.17	5.0	5.0	5.0
Acc%	57.4	61.2	70.2	69.6	69.6	87.5	89.8	96.0	96.0	96.8	96.5	97.3	97.3	97.3	97.3	97.3	97.3	97.3

#f.	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
TP%	95.0	95.0	95.0	94.8	95.0	94.8	95.0	94.8	94.8	94.8	94.8	94.8	94.7	94.7	94.7	94.7	93.8
FP%	0.17	0.33	0.17	0.50	0.67	0.50	0.50	0.33	0.17	0.33	0.33	0.33	0.17	0.33	0.50	0.67	0.33
TN%	99.8	99.7	99.8	99.5	99.3	99.5	99.5	99.7	99.8	99.7	99.7	99.7	99.8	99.7	99.5	99.3	99.7
FN%	5.00	5.00	5.00	5.17	5.00	5.17	5.00	5.17	5.17	5.17	5.17	5.17	5.33	5.33	5.33	5.33	6.17
Acc%	97.4	97.3	97.4	97.2	97.2	97.2	97.3	97.3	97.3	97.3	97.3	97.3	97.3	97.2	97.1	97.0	96.8

Table 5
Feature numbers vs. performances for TestB.

#f.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
TP%	28.7	29.7	29.7	50.3	50.3	50.3	57.5	57.5	57.5	57.5	57.5	57.5	57.5	57.5	57.5	56.7	56.7	56.7
FP%	0.00	1.67	1.67	2.50	2.50	2.50	3.67	3.67	3.67	3.67	3.67	3.67	3.67	3.67	3.67	2.00	2.17	2.17
TN%	100.	98.3	98.3	97.5	97.5	97.5	96.3	96.3	96.3	96.3	96.3	96.3	96.3	96.3	96.3	98.0	97.8	97.8
FN%	71.3	70.3	70.3	49.7	49.7	49.7	42.5	42.5	42.5	42.5	42.5	42.5	42.5	42.5	42.5	43.3	43.3	43.3
Acc%	64.3	64.0	64.0	73.9	73.9	73.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	77.3	77.3	77.3

#f.	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
TP%	56.3	54.8	55.0	56.3	54.0	56.7	56.3	56.5	56.7	57.3	56.2	54.8	56.5	55.8	55.8	56.0	51.2
FP%	1.00	1.00	0.83	0.83	0.83	0.83	1.33	0.83	1.33	1.33	2.50	0.83	2.33	1.00	1.00	1.50	1.00
TN%	99.0	99.0	99.2	99.2	99.2	99.2	98.7	99.2	98.7	98.7	97.5	99.2	97.7	99.0	99.0	98.5	99.0
FN%	43.7	45.2	45.0	43.7	46.0	43.3	43.7	43.5	43.3	42.7	43.8	45.2	43.5	44.2	44.2	44.0	48.8
Acc%	77.7	76.9	77.1	77.8	76.6	77.9	77.5	77.8	77.7	78.0	76.8	77.0	77.1	77.4	77.4	77.3	75.1

overall accuracy was 96.75%. In Fig. 8, the best accuracy was 78% as 28 features were applied. Again, as all 35 features were applied in Table 3 the result was slightly lowered to be 75.08%. The numeric values in details for trained weight vectors in Figs. 7 and 8 are given in Tables 4 and 5, respectively.

It is interesting to note that, as the number of applied features is larger than a specific number, or is smaller than another specific number, the performances of weighted vectors are significantly superior to that of untrained vectors. This may be because some features are not necessary to be considered in the KNN classification. On the other hand, as the features are cut too much, the importance of our weighting emerges. We also note that the training time for computing an optimal chromosome in a contemporary personal computer was several minutes in the case of 100 generations and 30 initial chromosomes, with 1200 instances in the training set and 200 instances in the sampling set. Once the optimal chromosome was obtained in the training stage, one tested instance can be classified by KNN in tens of milliseconds. Therefore, the proposed method can be surely applied to a real-time system.

5. Conclusion

This study proposed a method to weight features of DoS/DDoS attacks, and analyzed the relationship between detection performance and number of features. Compared to previous works, the proposed method can be implemented to a real-time NIDS, since all features applied in this study were directly collected from packet headers. All 35 features were evaluated and weighted. According to the experiments on DoS/DDoS attacks, (1) for known attack detection, the best overall accuracy was 97.42%, for which only top 19 features were considered; (2) for unknown attack detection, the best overall accuracy was 78%, for which top 28 features were considered.

Acknowledgments

This work was partially supported by the National Science Council with contracts NSC 95-2221-E-130-003 and 96-2221-E-130-009.

References

Abbes, T., Bouhoula, A., & Rusinowitch, M. (2004). Protocol analysis in intrusion detection using decision tree. In *Proceedings of the IEEE conference on information technology: coding and computing* (pp. 404–409).

Carl, G., Kesidis, G., Brooks, R. R., & Rai, S. (2006). Denial-of-service attack-detection techniques. *IEEE Internet Computing*(January–February), 82–89.

DARPA Intrusion Detection Evaluation, <http://www.ll.mit.edu/IST/ideval/data/data_index.html>.

Gavrilis, D., & Dermatas, E. (2005). Real-time detection of distributed denial-of-service attacks using rbf networks and statistical features. *Computer Networks, 48*(2), 235–245.

Hofman, A., Horeis, T., & Sick, B. (2004). Feature selection for intrusion detection: an evolutionary wrapper approach. In *Proceedings of the IEEE neural networks* (Vol. 2, pp. 1563–1568).

Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press.

IP Traffic. <http://www.omnicor.com/netest.htm>.

KDD CUP, (1999a). Datasets <http://www.sigkdd.org/kddcup/index.php?section=1999&method=data>.

KDD CUP, (1999b). A list of features, <http://www.sigkdd.org/kddcup/site/1999/files/kddcup.names>.

Kulkarni, A., & Bush, S. (2006). Detecting distributed denial-of-service attacks using Kolmogorov complexity metrics. *Journal of Network and Systems Management, 14*(1), 69–80.

Lee, C. H., Chung, J. W., & Shin, S. W (2006). Network intrusion detection through genetic feature selection. In *Proceedings of the IEEE conference on software engineering, artifical intelligence, networking, and parallel/distributed computing (SNPD)* (pp. 109–114).

Liao, Y., & Rao Vemuri, V. (2002). Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security, 21*(5), 439–448.

Middlemiss, M. J., & Dick, G. (2003). Weighted feature extraction using a genetic algorithm for intrusion detection. In *Proceedings of the evolutionary computation* (Vol. 3, pp. 1669–1675).

Mukkamala, S., & Sung, A. H. (2002). Feature ranking and selection for intrusion detection using support vector machines. In *Proceedings of the conference on information and knowledge engineering* (pp. 503–509).

Stein, G., Chen, B., Wu, A. S., & Hua, K. A. (2005). Decision tree classifier for network intrusion detection with GA-based feature selection. In *Proceedings of the ACM southeast regional conference* (Vol. 2, pp. 136–141).

Sung, A. H., & Mukkamala, S. (2003) Identifying important features for intrusion detection using support vector machines and neural networks. In *Proceedings of the IEEE symposium on applications and the Internet* (pp. 209–216).

Wang, H., Zhang, D., & Shin, K. G. (2004). Change-point monitoring for the detection of DoS attacks. *IEEE Transactions on Dependable and Secure Computing, 1*(4), 193–208.