IEICE TRANS. INF. & SYST., VOL.E97–D, NO.8 AUGUST 2014

PAPER

Unsupervised Learning Model for Real-Time Anomaly Detection in Computer Networks

Kriangkrai LIMTHONG[†a)], *Nonmember*, **Kensuke FUKUDA**[††], **Yusheng JI**[††], *and* **Shigeki YAMADA**[††], *Members*

SUMMARY Detecting a variety of anomalies caused by attacks or accidents in computer networks has been one of the real challenges for both researchers and network operators. An effective technique that could quickly and accurately detect a wide range of anomalies would be able to prevent serious consequences for system security or reliability. In this article, we characterize detection techniques on the basis of learning models and propose an unsupervised learning model for real-time anomaly detection in computer networks. We also conducted a series of experiments to examine capabilities of the proposed model by employing three well-known machine learning algorithms, namely multivariate normal distribution, *k*-nearest neighbor, and one-class support vector machine. The results of these experiments on real network traffic suggest that the proposed model is a promising solution and has a number of flexible capabilities to detect several types of anomalies in real time.

key words: *machine learning, multivariate normal distribution, nearest neighbor, one-class support vector machine*

1. Introduction

A study by Hansman and Hunt [1] shows that over the past decades, computer attacks have shown a marked increase in the usage of sophisticated techniques to evade existing intrusion detection systems. Another type of anomaly caused by accidents, such as outages or misconfigurations, has also had adverse effects on availability and reliability of computer systems. However, it is quite difficult for network operators to inspect every single network packet or flow for these types of anomalies because the volume of Internet traffic has been increasing exponentially as well. Thus, the need to automatically detect attacks and unusual incidents in computer networks is of crucial importance. Please note that, intrusion detection is a subset of anomaly detection in our context.

Extensive surveys of anomaly detection techniques by Patcha and Park [2], and Chandola *et al.* [3] show that existing techniques are conventionally classified into two main categories: signature detection and anomaly detection techniques. Additionally, a recent study by Tsai *et al.* [4] suggests that machine learning techniques can play a major role in anomaly detection in computer networks. Several learning algorithms, such as *k*-nearest neighbor algorithms [5], neural networks [6], and support vector machines [6], [7], have been widely proposed and applied to network traffic detection. Nevertheless, nearly all of the previous studies utilize batch processing, which requires a sufficient amount of input data before detecting anomalies, and thus these techniques would not be practical solutions for detecting anomalies in real-time.

There are several issues that make anomaly detection in computer networks more difficult and different from other areas. The first issue is that classifying data as either normal or anomaly in computer networks depends on a variety of factors in a particular situation. For example, assume we collected data from a small office; if the data shows someone surfs the Internet from inside during office hours, we easily classify the data of this behavior as normal. However, if exactly the same data appear at midnight when nobody was working there, and this behavior had never happened before, it is quite difficult to clearly define the data of this behavior as normal or anomaly. From this example, classifying data depends on conditions at a particular time in a particular place. The second issue is that attackers place a large amount of effort into imitating data to evade or influence detection systems in computer networks, but such imitation rarely occurs in other areas. The last issue is that it seems easy to acquire completely labeled data in other areas, but it is very costly to obtain such data from real computer networks to train classifiers. Thus, the real challenge of our study is not only expeditiously detecting a wide range of anomalies but also developing a robust technique that is not easily influenced by attackers or suffers from unlabeled training data.

The objective of our study is to make the following contributions:

1. To characterize and analyse learning models for anomaly detection in computer networks, and propose a learning model for real-time anomaly detection.
2. To evaluate detection performance of learning algorithms with the proposed model for several types of anomalies.
3. To test robustness of learning algorithms with the proposed model to incorrect training data.
4. To investigate the learning curves of learning algorithms with the proposed model.
5. To measure time consumption of algorithms with the proposed model in the learning and detecting process.

The remainder of this paper is structured as follows: In Sect. 2, we characterize and analyse existing learning mod-

Manuscript received November 11, 2013.
Manuscript revised March 11, 2014.

[†]The author is with the Graduate University of Advanced Studies (Sokendai), Tokyo, 101–8430 Japan.
[††]The authors are with the National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: krngkr@nii.ac.jp

DOI: 10.1587/transinf.E97.D.2084

els for anomaly detection in computer networks, and highlight key problems of each model. We then describe and explain our proposed model. In Sect. 3, we explain how to acquire and prepare experimental data, how to conduct experiments and how to evaluate detection performance. Next, we focus on each particular experiment and show results in Sect. 4. We discuss the experimental results and describe the steps required to apply our model to real network environments in Sect. 5. Finally, we draw our conclusions in Sect. 6.

2. Related Work and Our Proposed Model

In this section, we focus on learning models for anomaly detection in computer networks and raise several important issues of these models. We then describe our proposed learning model and highlight some advantages. Please note that we mainly concentrate on the learning and detecting processes rather than post-processing after the anomalies have been detected.

To simplify the explanation for learning models in this section, we assume that a data instance occurs in every interval on a timeline for one day. An instance at time $t = x$ is represented by $\mathbf{x}$, and when we consider data on multiple timelines, we represent an instance $\mathbf{x}$ on the present timeline p with $\mathbf{x}^p$, and so on. The ultimate goal of anomaly detection is identifying a function $f(\mathbf{x})$ that can classify an instance $\mathbf{x}$ into either the normal class or anomaly class. Therefore, we can define the task of anomaly detection as a binary classification problem [8], and we can evaluate performance of $f(\mathbf{x})$ with measurements for binary classification problems.

A simple way to create the classification function $f(\mathbf{x})$ is to let the function be manually specified by experts; we call this technique as "manual-based model". Figure 1 (a) depicts this model and shows a detecting connection (a bold line) between the instance $\mathbf{x}$ and classification function $f(\mathbf{x})$. Examples of "manual-based model" are firewall systems, anti-virus software, and signature-based intrusion detection. Many commercial products also conform to this model including Snort [9], Bro [10], NetSTAT [11], RealSecure, and Cisco Secure IDS. Despite its popularity, this model is not a flexible solution and obviously cannot detect novel and unknown anomalies that are not defined in the classification function.

To automatically generate the classification function $f(\mathbf{x})$, some studies have proposed what we call the "batch model", as shown in Fig. 1 (b). Let us assume that instances occur at $t = v$, w, x, y, and z, a single instance per interval sequentially. Generally, a learning algorithm of this model generates $f(\mathbf{x})$ by learning from all instances except the test instance, for example from $\mathbf{v}$, $\mathbf{w}$, $\mathbf{y}$, and $\mathbf{z}$ represented by learning connections (dash lines). Many algorithms from recent studies learn from an entire data set [12] or require a certain amount of data [13] before detecting an individual or group of test instances are classified under "batch model" as well. However, this model is more suitable for an offline system rather than for a real-time system because in a real-time system we cannot acquire any information from instances that occur after the present instance.

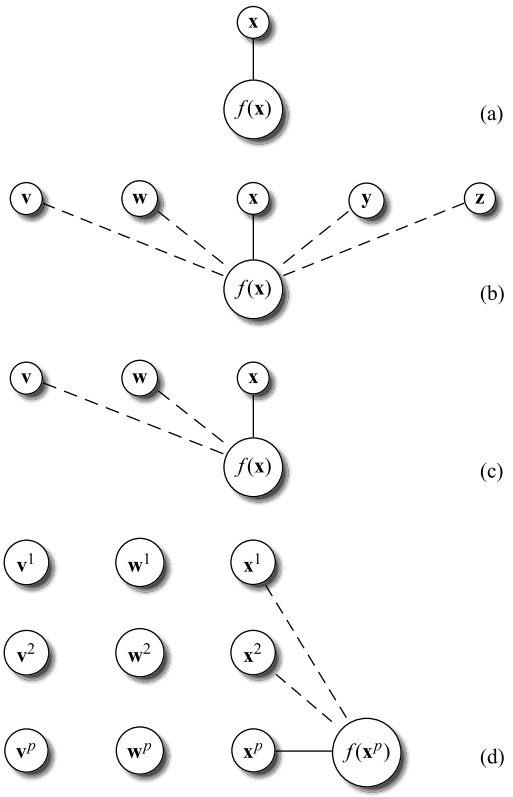

Fig. 1 Graphical models for anomaly detection: (a) manual-based model, (b) batch model, (c) real-time model, and (d) the proposed model with learning connection (dashed line) and detecting connection (bold line).

Figure 1 (c) shows a model which is more practical for a real-time system called the "real-time model". A learning algorithm of the model generates the classification function by learning from data that occurs before the present instance $\mathbf{x}$, learning from $\mathbf{v}$ and $\mathbf{w}$ for example [14], [15]. This model has been used by real-time detection systems in many other areas, but there are three key points under our consideration for applying the model to computer networks. The first point is how to classify the present instance when there is no prior data, instance $\mathbf{v}$ for example. The second point is what amount of prior data is sufficient to generate the classification function. The last point is that the data from this model are easily imitated or manipulated by attackers to evade detection systems, because prior data have a strong influence on the classification function for the present instance.

To improve the "real-time model", we propose our model, as shown in Fig. 1 (d). An algorithm of the proposed model learns from data on preceding timelines, when each timeline has the same length. Figure 1 (d) shows three timelines for example, timeline number 1, 2, and p which indicates the present timeline. A learning algorithm generates the classification function $f(\mathbf{x})$ by learning from the prior data at the present time stamp on preceding timelines. For example, from Figure 1 (d) assume that we would like to classify a test instance at time stamp $t = x$ on the present

timeline, represented by $\mathbf{x}^p$. The algorithm generates the classification function for $\mathbf{x}^p$ by learning from prior data at time stamp $t = x$ on the timeline number 1 and 2, which are represented by $\mathbf{x}^1$ and $\mathbf{x}^2$, respectively.

Changing the learning model from horizontal to vertical axis, as our proposed model does, results in several advantages. The first advantage is that it is quite difficult for attackers to imitate or manipulate data to evade detection system when we set the distance between timelines to be sufficiently long. Figure 1 (d) for example, assume each timeline is one day long, such that the distance between $\mathbf{x}^p$ and $\mathbf{x}^2$ is one day, and the distance between $\mathbf{x}^p$ and $\mathbf{x}^1$ is two days. It seems impossible for attackers to go back and manipulate data from yesterday or two days ago. The second advantage is that even if attackers could imitate data on the present timeline, this action has no influence on the classification function for the following instances. In Fig. 1 (d), for example, attackers might create imitated instance $\mathbf{v}^p$ and $\mathbf{w}^p$ but does not influence the classification function for instance $\mathbf{x}^p$, because $f(\mathbf{x}^p)$ is learned from $\mathbf{x}^1$ and $\mathbf{x}^2$ that occur at $t = x$ on preceding timelines. The last advantage is our analysis suggests that the proposed model, with appropriate learning algorithms, tolerates incorrect training data because there is a very low possibility that incorrect data can occur at the same time stamp on different timelines. Therefore, to confirm our analysis, we conducted a series of experiments to examine these key aspects of our proposed model.

3. Materials and Methods

In this section, we explain about data sources and preparation, data representation and preprocessing steps, learning algorithms, and performance evaluation.

3.1 Data Sources and Preparation

We acquired data from two different sources, one for normal traffic and the other for abnormal traffic. The normal traffic is from an edge router of the Internet service center in Kasetsart University, Thailand. This center provides computer clients for all members to access the Internet and various local services. There are approximately 1,300 users every day among 157 computer clients, and the service hours are between 8:30 and 24:00 on weekdays. Anti-virus and appropriate software for ordinary users are installed on each client, and users cannot modify or install any other software. We collected traffic data from this source for 3 months and then manually selected 55 days according to official statistics that indicate normal behavior of usage. For example, official statistics indicates low usage over midterm or final examination periods, so we did exclude network traffic during midterm and final examination. Consequently, we assume that all traffic data from this source are normal.

For abnormal traffic, we extracted attack packets from a test bed of the DARPA Lincoln Lab in Massachusetts Institute of Technology [16], [17]. This data set was mainly provided for evaluation of intrusion detection systems. We selected five types of attacks from the data set as follows:

1. *Back* attack is a denial of service attack against the Apache web server through port 80, where a client requests a URL containing many backslashes.
2. *IpSweep* attack is a surveillance sweep performing either a port sweep or ping on multiple IP addresses.
3. *Neptune* attack is a SYN flood denial of service attack on one or more destination ports.
4. *PortSweep* attack is a surveillance sweep through many ports to determine which services are supported on a single host.
5. *Smurf* attack is an amplification attack using ICMP echo reply flood.

Essential characteristics of the selected attacks are listed in Table 1. In the first column, we indicate sources and types of anomalies for each instance. In the next five

Table 1 Characteristics of selected attacks.

Source	No. of SrcAddr	No. of DstAddr	No. of SrcPort	No. of DstPort	No. of Packet	Average Packet Size (Byte)	Duration (sec.)	%Anomaly
Back								
Week 2 Fri	1	1	1,013	1	43,724	1,292.31	651	0.75
Week 3 Wed	1	1	999	1	43,535	1,297.29	1,064	1.23
IpSweep								
Week 3 Wed	1	2,816	1	104	5,657	60.26	132	0.15
Week 6 Thu	5	1,779	2	105	5,279	67.75	4,575	5.30
Neptune								
Week 5 Thu	2	1	26,547	1,024	205,457	60	3,143	3.64
Week 6 Thu	2	1	48,932	1,024	460,780	60	6,376	7.38
Week 7 Fri	2	1	25,749	1,024	205,600	60	3,126	3.62
PortSweep								
Week 5 Tue	1	1	1	1,024	1,040	60	1,024	1.19
Week 5 Thu	1	1	1	1,015	1,031	60	1,015	1.17
Week 6 Thu	2	2	2	1,024	1,608	60	1,029	1.19
Smurf								
Week 5 Mon	7,428	1	1	1	1,931,272	1,066	1,868	2.16
Week 5 Thu	7,428	1	1	1	1,932,325	1,066	1,916	2.22
Week 6 Thu	7,428	1	1	1	1,498,073	1,066	1,747	2.02

columns, we show primitive characteristics of each anomaly instance: the number of source addresses, destination addresses, source ports, destination ports, and packets. Next, the average packet size and duration of each anomaly instance are shown in the seventh and eighth columns. Finally, the percentage of each instance in one day is shown in the last column.

Concerning data preparation for the experiments, we divided the 55-day normal traffic into a 39-day (≈70%) traffic data set and a 16-day (≈30%) traffic data set. We used purely the 39-day traffic data as a training set without any modification. On the other hand, we produced a test set for individual types of attacks by combining the 16-day traffic data with instances of each attack. For example, we combined the 16-day traffic data with two instances of the *Back* attack, as listed in Table 1, to produce a 32-day test set for the *Back* attack, and so forth. Therefore, we have a 32-day test set for the *Back* and *IpSweep*, and a 48-day test set for the *Neptune*, *PortSweep*, and *Smurf* attacks separately.

3.2 Data Representation

We represented an instance of network traffic as an f-dimensional vector known as a feature vector, where f is the number of features be used. In our experiments, we would like to detect the time when anomalies occur, so we presented an interval as a feature vector. However, a single interval may comprise a number of network packets; thus, we aggregated all packets in a particular interval. We then extracted features from a single interval and represented an instance $\mathbf{x}$ at time $t = x$ in f-dimensional space, with

$$\mathbf{x} = \left[x_1 \dots x_f\right]^T \in \mathbb{R}^f, \tag{1}$$

where $x_1 \dots x_f$ are scalar-valued random variables of features, T denotes a transposition of feature vector, and f is the number of features. However, we extracted nine features from network traffic and then performed feature scaling to values between 0 and 1 (described in the following section) such that we can rewrite Eq. 1 with

$$\mathbf{x} = \left[f_1 \dots f_9\right]^T \in \mathbb{R}^9_{[0,1]}. \tag{2}$$

3.3 Feature Extraction and Feature Scaling

We have two steps involved with the features of network data: feature extraction and feature scaling.

For feature extraction, we extracted nine features as listed in Table 2 during packet aggregation for each interval. In this table, we listed the abbreviation, feature name, and description of each feature. Please note that the ΔAddr (f_8) is a difference in the number of addresses between source and destination, and the ΔPort (f_9) is a difference in the number of ports between source and destination. All of the features are selected to account for characteristics of the selected attacks listed in Table 1.

For feature scaling, we normalized the wide range of different features into a standard range of 0 to 1. We scaled features according to

$$\hat{x}_{i,j} = \frac{x_{i,j}}{\max_j(x_{i,j})}, \forall_{i=1\dots f} \wedge \forall_{j=1\dots m}, \tag{3}$$

where $\hat{x}_{i,j}$ is a scaled feature, $\max_j(x_{i,j})$ is the maximum value of the data in the i-th feature, m is the number of samples in the training data, and f is the number of features.

Table 2 Features of network traffic on an interval basis.

f#	Feature	Description
f_1	Packet	Number of packets
f_2	Byte	Sum of packet size
f_3	Flow	Number of flows
f_4	SrcAddr	Number of source addresses
f_5	DstAddr	Number of destination addresses
f_6	SrcPort	Number of source ports
f_7	DstPort	Number of destination ports
f_8	ΔAddr	|SrcAddr − DstAddr|
f_9	ΔPort	|SrcPort − DstPort|

3.4 Learning Algorithms

We employed three well-known algorithms of machine learning [18], namely the multivariate normal distribution, k-nearest neighbor, and one-class support vector machine, to work with our proposed model.

3.4.1 Multivariate Normal Distribution (MND)

The MND is a generalization of the Gaussian or normal probability density function in high dimensions [19]. In the f-dimensional space, the probability density function is given by

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{f/2}\,|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{4}$$

where $\boldsymbol{\mu} = E[\mathbf{x}]$ is the vector of mean value, and $\mathbf{\Sigma}$ is the $f \times f$ covariance matrix defined as

$$\mathbf{\Sigma} = E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T], \tag{5}$$

where $|\mathbf{\Sigma}|$ denotes the determinant of $\mathbf{\Sigma}$.

To classify test data, we defined an adaptive threshold

$$\varepsilon = \frac{1}{(2\pi)^{f/2}\,|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}\rho^2\right), \tag{6}$$

where ρ is a parameter to obtain the proportion of maximum probability. Smaller values of ρ produce higher probabilities than larger values. We varied ρ between 2 and 4 on a linear scale to determine the best detection performance. We defined the classification function of test data $\mathbf{x}$ as

$$f(\mathbf{x}) = \begin{cases} \text{anomaly} & \text{if } p(\mathbf{x}) < \varepsilon, \\ \text{normal} & \text{otherwise.} \end{cases} \tag{7}$$

3.4.2 K-nearest Neighbor (KNN)

The KNN is an instance-based learning algorithm for classifying data points based on the closest learning samples in

the f-dimensional space [20]. In our experiments, the nearest neighbors of the data are defined by standard Euclidean distances. More precisely, let test data $\mathbf{x}$ comprising f features be described by the feature vector $(x_1 \ldots x_f)$, where x_i denotes the value of the i-th feature of data $\mathbf{x}$. The Euclidean distance between two instances $\mathbf{x}$ and $\mathbf{y}$ is defined by

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{f} (x_i - y_i)^2}. \tag{8}$$

To classify test data, we specified the constant parameter $k = 3$ and defined the classification function of test data $\mathbf{x}$ as

$$f(\mathbf{x}) = \begin{cases} \text{anomaly} & \text{if amount of training data nearest} \\ & \text{to } \mathbf{x} \text{ is less than } k \text{ in the distance } D, \\ \text{normal} & \text{otherwise.} \end{cases} \tag{9}$$

Thanks to the feature scaling step, we can vary the parameter D on a logarithmic scale between 10^{-6} and 10^{0} for selection of the best detection performance.

3.4.3 One-Class Support Vector Machine (OSVM)

The OSVM introduced by Schölkopf *et al.* [21] is a variation of the standard support vector machines (SVM) algorithm. The main idea is that the OSVM maps unlabeled input data into a high dimensional space via an appropriate kernel function and then attempts to find hyperplanes that separate the input data with maximum margin.

To classify test data, we defined the decision function according to Muller *et al.* [22],

$$h(\mathbf{x}) = \text{sign}((\boldsymbol{\omega}.\Phi(\mathbf{x})) - \rho) \tag{10}$$

will be positive for most samples $\mathbf{x}_i$ contained in the training set and negative for the opposite, where $\boldsymbol{\omega}$ is a normal vector of hyperplane, Φ is a kernel function, and ρ represents a penalized value in the object function. Therefore, we defined the classification function of test data $\mathbf{x}$ as

$$f(\mathbf{x}) = \begin{cases} \text{anomaly} & \text{if } h(\mathbf{x}) = -1, \\ \text{normal} & \text{if } h(\mathbf{x}) = +1. \end{cases} \tag{11}$$

In our experiments, we used the LIBSVM [23] tool with a radial basis function (RBF) as an appropriate kernel. We used the standard parameters of this tool for all experiments; however, we varied the *nu* and *gamma* parameters (standard parameters for RBF) on a logarithmic scale between 10^{-3} and 10^{0} for selection of the best detection performance.

3.5 Performance Evaluation

To evaluate detection performance we used the F-score [24] as a single measure, which considers both the precision and recall [25] to compute the score. We measured the precision, recall, and F-score based on entire intervals. All of these values are derived from the parameters as listed in Table 3: the true positive (TP), which is the number of anomalous intervals correctly detected; the false positive (FP), which is the number of normal intervals wrongly detected as anomalous intervals; the false negative (FN), which is the number of anomalous intervals not detected; and the true negative (TN), which is the number of normal intervals correctly detected. From these four parameters, the precision, recall, and F-score are calculated by

Table 3 Confusion matrix for anomaly detection.

Detected Class	Actual Class	
	Anomaly	Normal
Anomaly	True Positive (TP)	False Positive (FP)
Normal	False Negative (FN)	True Negative (TN)

$$precision = \frac{TP}{TP + FP}, \tag{12}$$

$$recall = \frac{TP}{TP + FN}, \tag{13}$$

$$F\text{-}score = 2 \times \frac{precision \times recall}{precision + recall}. \tag{14}$$

The precision (positive predictive value), Eq. (12), is the percentage of detected intervals that are actually anomalies. The recall (sensitivity value), Eq. (13), is the percentage of the actual anomalous intervals that are detected. The F-score, Eq. (14), is the harmonic mean of the precision and recall. The F-score is in the range of 0 to 1, where 0 represents the worst detection rate, and 1 represents the perfect detection rate.

4. Results

In this section, we explain the purpose and procedure for conducting each experiment. We segmented our study into five experiments as follows.

4.1 Experiment 1: Comparison of Different Interval Values

An objective of this experiment is to identify the practical interval value for the experimental data. We varied interval values with 1, 10, 20, 30, 40, 50, and 60 seconds. In this experiment, we performed anomaly detection with individual features and individual algorithms, then computed the F-score for each type of attack. The results of this experiment are shown in Fig. 2.

We provide a full description of Fig. 2 with the following details. The y-axis shows the average F-score, where a higher F-score is a better detection performance than a lower F-score, and the x-axis indicates interval values from 1 to 60 seconds. From the first to third row, we respectively show the detection performances of MND, KNN and OSVM algorithms over each type of attack. The bottom row shows the average detection performance of each algorithm for all

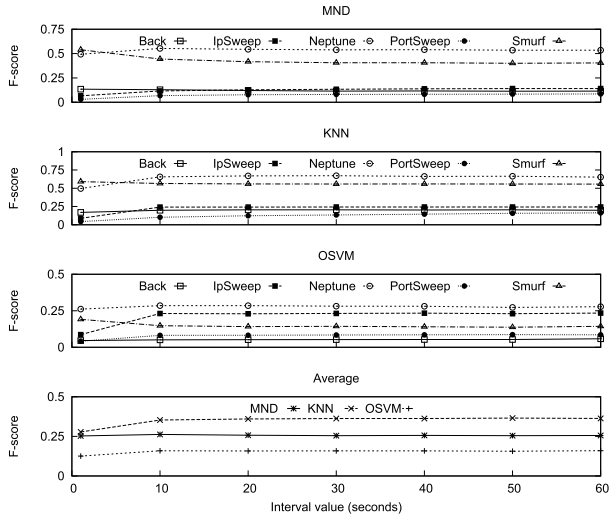

Fig. 2 Detection performance with different interval values using MND, KNN, and OSVM algorithms.

types of attacks by altering the interval value.

The results suggest that the practical interval value for our experimental data is 10 seconds. For individual algorithms, we found that most detection performances increased from 1 to 10 seconds, especially in *IpSweep* and *Neptune* attacks; however, the detection performance in *Smurf* attacks decreases from 1 to 10 seconds for all three algorithms. The detection performances of interval values between 20 and 60 seconds for all algorithms are slightly different from those of 10 seconds; therefore, we assigned 10 seconds to the interval value for the rest of experiments.

4.2 Experiment 2: Performance Comparison between Individual Features and Combined All Features

A purpose of the second experiment is to investigate feasible features for each type of attack. We compared detection performance using individual features and also using a combination of all nine features. We conducted this experiment with the following steps. First, every algorithm learned from the same training data using a single feature one by one and then using the combined feature. Next, we evaluated performance on individual test data for each type of attack using the same feature as learning process. We present detection performances of the individual features and the combined feature in a bar graph as shown in Fig. 3.

The figure shows comparison of detection performance with the following details. The y-axis on the graph represents the F-score in the range of 0 to 1, where 0 indicates the worst detection rate and 1 indicates the perfect detection rate. The x-axis indicates the individual features ($f_1 - f_9$), as listed in Table 2, and the combined all nine features (f_{all}). The different vertical bars represent detection performances using MND, KNN, and OSVM algorithms, respectively. To compare results with another model, we also show detection performances of the real-time model as RTM bars in this figure. The RTM bar represents the best detection performance from MND, KNN, and OSVM algorithms by using real-time model as explained in Sect. 2. Each row indicates results for different types of attacks put in order by *Back*, *IpSweep*, *Neptune*, *PortSweep*, and *Smurf* attacks.

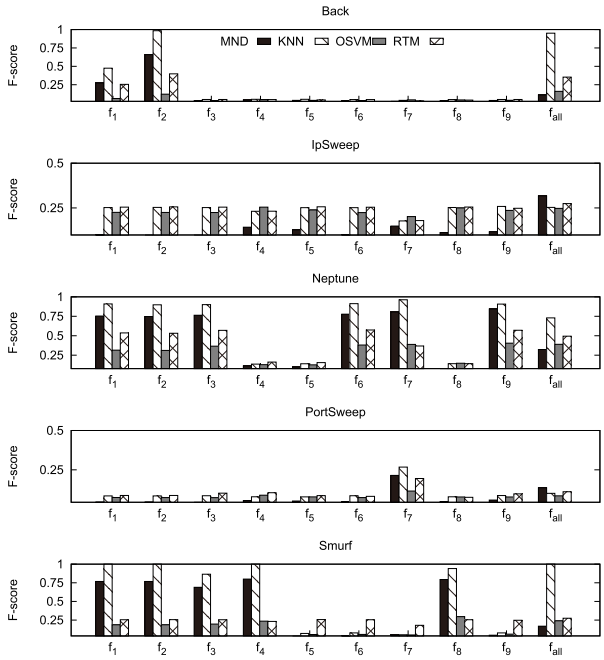

Fig. 3 Performance comparison between individual features ($f_1 - f_9$) and combined all features (f_{all}) for each type of attack using MND, KNN, OSVM algorithms, and real-time model.

The results from this experiment show the following points. For the MND algorithm, using an individual feature for a particular attack produced higher performance than using a combined feature, except for *IpSweep* attacks. For the KNN and OSVM algorithms, however, detection performance slightly dropped when using a combined feature compared to the obtained results using only individual features. Detection performance of the proposed model is better than those of real-time model (RTM), specifically when compared to MND and KNN. Obviously, effective features of real-time model are the same features as those by using the proposed model. We cannot compare the proposed model to the manual-based model and batch model because of their limitations. The manual-based model cannot detect anomalies which are not contained in database or discriminant function and batch model cannot be applied in real time, for example.

4.3 Experiment 3: No Packet Situations

We divided this experiment into two sub-experiments on account of two different purposes: the first is to investigate how learning algorithms with the proposed model detects anomalies caused by an accident, and the other is to evaluate the robustness of the proposed model with incorrect training data.

For the first purpose, we simulated that an outage oc-

Table 4 Percentage difference in detection performance between the original training data and adding no packet to the training data.

Attack	MND			KNN			OSVM		
	1 day	3 days	5 days	1 day	3 days	5 days	1 day	3 days	5 days
Back	−0.10%	−0.54%	−0.61%	+0.15%	+0.41%	+0.41%	−0.36%	+1.33%	+1.73%
IpSweep	−2.90%	−5.47%	−5.83%	−0.01%	+0.01%	+0.01%	−1.12%	−3.14%	−0.32%
Neptune	+0.60%	−0.50%	−1.16%	+0.40%	+0.86%	+0.86%	−1.05%	+6.93%	+7.59%
PortSweep	−1.07%	−2.37%	−2.60%	+0.06%	+0.20%	+0.20%	−0.80%	−0.82%	−0.42%
Smurf	+0.40%	+0.38%	+0.33%	+0.01%	+0.04%	+0.04%	−1.78%	+4.25%	+6.86%

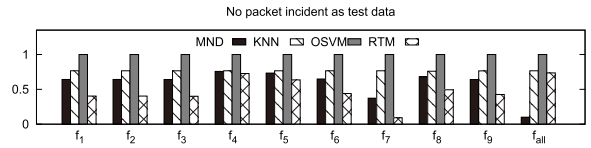

Fig. 4 Detection performance of one-day long with no packet incident using MND, KNN, OSVM algorithms, and real-time model.

curs, and thus, there were no packets at all for an entire day. We then attempted to detect this unusual incident in test data. We show the results in Fig. 4 with the same details as the results of experiment 2 in Sect. 4.2. The y-axis shows the F-score, and the x-axis indicates the individual features ($f_1 - f_9$) and the combined feature (f_{all}). The different vertical bars represent detection performance of MND, KNN, and OSVM algorithms, respectively. The last vertical bar, RTM, represents detection performance of real-time model.

Figure 4 shows that the detection performance from high to low rates are OSVM, KNN, and MND algorithms respectively. All detection performances of learning algorithms for the combined feature are the same sequential as those for each individual feature. Most former detection techniques, however, cannot detect no packets for entire day like this situation. Detection performance of real-time model (RTM) are the worst of all by using individual features; however, detection performance of RTM seems close to KNN by using the combined feature. It seems reasonable to assume that this situation is similar to data imitation or manipulation to influence the classification function from attackers. The real-time model has been adversely affected by situations like this, so it is a major reason why detection performance of real-time model is the lowest. However, the combined feature makes the real-time model more robust for detecting anomaly in this situation. It means that the proposed model would probably detect anomalies caused by accidents, and tolerate data imitation or manipulation from attackers rather than the real-time model.

For the second purpose, we assumed that we accidentally included data with no packets for an entire day into the training data. We then attempted to detect individual types of anomaly with the same procedure as the experiment 2 in Sect. 4.2. Afterwards, we measured the percentage difference of detection performance between this experiment and the former experiment, as listed in Table 4. From the table, the first column shows types of attacks, and we group the rest of columns into each of the algorithms. We show the percentage difference in detection performance with 1, 3, and 5 days of no packet in the training data, where (+) indicates the performance increase, and (−) indicates the performance decrease from the experiment 2 in Sect. 4.2.

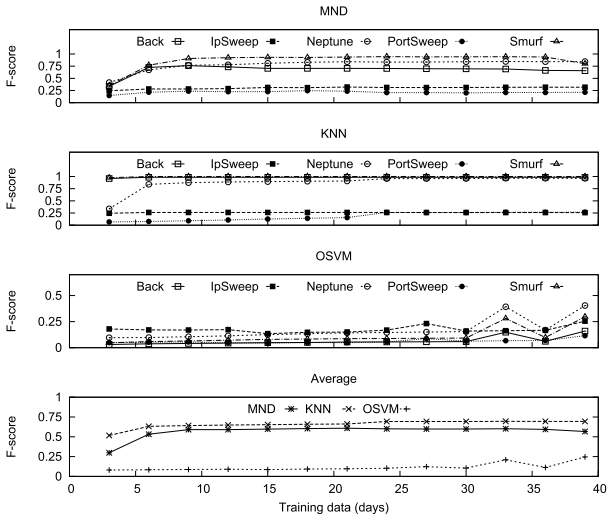

Fig. 5 Learning curves of MND, KNN, and OSVM algorithms with different amounts of training data.

The results as listed in Table 4 show that there are slight differences between detection performance of original training data and those of adding a no packet incident to the training data. Most detection performances of the proposed model using MND algorithm steady decreased, but using KNN and OSVM randomly changed, when we added more no packet into the training data. The results also show the proposed model with KNN algorithm is the most robustness to incorrect training data followed by MND and OSVM algorithms respectively.

4.4 Experiment 4: Learning Curve

The aim of this experiment is to investigate how quickly three algorithms with the proposed model learn from training data. According to the experiment 2 in Sect. 4.2, all algorithms learned from the entire 39-day training data. In this experiment, however, we varied the number of training data from 3 to 39 days. We performed experiments on each type of attack using the best features discovered from the experiment 2 in Sect. 4.2.

We present the results in Fig. 5 with the following details. The y-axis represents the F-score from 0 to 1 and x-axis represents the number of training data. In first three

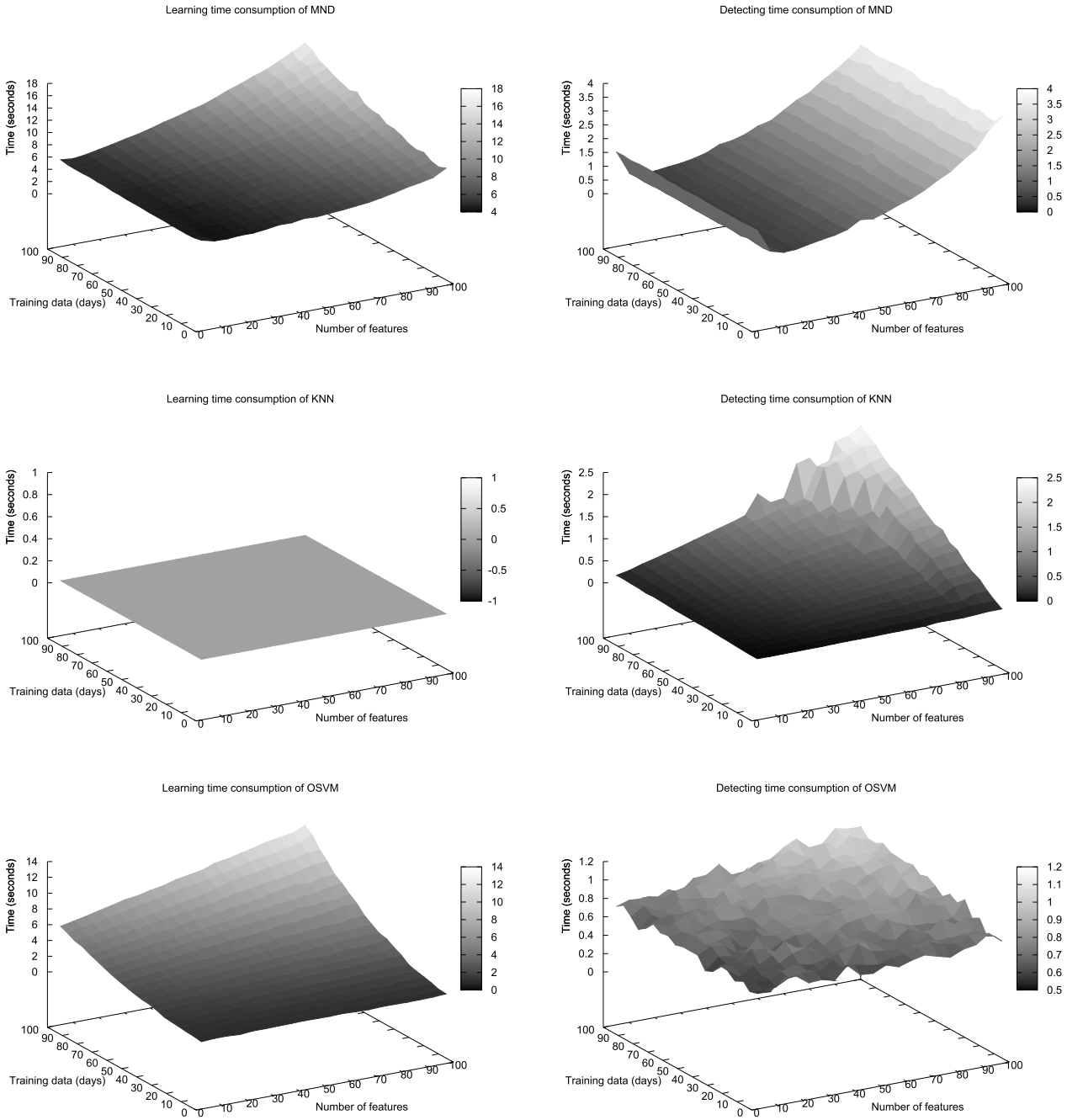

Fig. 6 Time consumption of learning process (left) and detecting process (right) for one-day test data using MND, KNN, and OSVM algorithms.

rows, we show the learning curve of each algorithm for separate types of attacks. The results in the bottom row show average learning curves of three learning algorithms.

We found that the MND and KNN algorithms learned very quickly from small amount of training data, and then each performance line of both algorithms converged to a constant value. Both of the algorithms required approximately nine days of training data then begin to converged. The OSVM algorithm, however, took a long period of time before detection performance began to increase, after 30 days of training data. We confidently expect that the detection performance of OSVM would be increased if we added more data in learning process.

4.5 Experiment 5: Time Consumption

The purpose of performing this experiment is to measure the time consumption of learning algorithms with the proposed model over both learning and detecting processes. We simulated experiment by varying the number of training data and the number of features from 1 to 100. Please note that we did not measure or include time consumption in the data

Table 5 Computational complexity of learning process and detecting process for one-day test data.

Algorithm	Learning Process	Detecting Process
MND	$O(mnf)$	$O(nf)$
KNN	$O(1)$	$O(mnf)$
OSVM	$O(m^2nf^2)$	$O(nfs)$

preprocessing steps, such as in feature extraction and feature scaling.

We show the time consumption of each algorithm in Fig. 6. The left-hand side shows time consumption of the learning process, and the right-hand side shows time consumption of the detecting process. The results from top to bottom are time consumption by using the MND, KNN, and OSVM algorithms, respectively. For all of the graphs, the x-axis indicates the number of features, the y-axis indicates the number of training data, and the z-axis shows time consumption.

All three learning algorithms have computational complexity as listed in Table 5, where m is the number of training data (days), n is the number of intervals in one day, f is the number of features, and s is the number of support vectors depending on pattern and number of training data. Our experimental results strongly support these time complexities. We also found that the proposed model could perform learning and detecting in real-time. For example, the maximum time consumptions when using the MND algorithm are approximately 18 and 4 seconds for learning and detecting process, respectively. In this experiment, we assigned the interval value to 10 seconds, and thus, n (the number of intervals in one day) is 8,640. Therefore, at a single interval, the MND algorithm spent $18 \div 8,640 \approx 2$ milliseconds for the learning process and $4 \div 8,640 \approx 0.46$ milliseconds for the detecting process. The KNN and OSVM algorithms also spent only few milliseconds in both learning and detecting process.

5. Discussion

From our experiments, we found that several factors had a major effect on detection performance of our purposed model. Two interrelated factors were a learning algorithm and selected features of network traffic. The results from experiments in Sect. 4.2 and the first part of Sect. 4.3 suggest that: (1) we have to select appropriate features, individual or combined features, for a particular type of attack when we employ the multivariate normal distribution (MND) algorithm with the proposed model; (2) we can use a single feature for a particular type of attack or a combined feature for all types of attacks when we employ the k-nearest neighbor (KNN) or one-class support vector machine (OSVM) algorithms with the proposed model, although detection performance of a combined feature slightly dropped in some cases.

The next factor in the proposed model that affected detection performance was the number of training days. The results from Sect. 4.4 show that the detection performance increased and converged to a constant value when we increased the number of training days, especially when using the MND and KNN algorithm. The learning curves of the OSVM algorithm, however, increased much slower than those by using MND and KNN. Interestingly, both the MND and KNN algorithms can learn from only one or two weeks of training data, whereas the OSVM algorithm requires more than one month of training data.

The last factor that affected detection was an interval value. The results from Sect. 4.1 show various performances when we assigned different interval values. Short interval values makes detection more likely to be a real-time system, but it take time more frequently for one-day computation, according to Table 5. Thus, this model is a trade-off between the benefits of a short time interval and resources consumption.

Experimental results strongly support our conclusions that the proposed model has a number of capabilities for real-time anomaly detection in computer networks. The proposed model provided flexibility in using various algorithms, features, and interval values to detect anomalies (Sects. 4.1, 4.2, 4.3). This model could also detect various anomalies caused by attacks and accidents (Sects. 4.2, 4.3). Additionally, the proposed model provides robustness to incorrect training data or data manipulation from attackers (Sect. 4.3). When we used the proposed model with appropriate learning algorithms, it could quickly learn from the training data to detect anomalies (Sect. 4.4). Interestingly, there was a high probability that the proposed model could apply for real-time anomaly detection (Sect. 4.5).

We found two interesting points from these experimental results. First, detection performances of OSVM for no packet incident as shown in Fig. 4 are much higher than those for selected attacks as shown in Fig. 3. The main reason is that the OSVM algorithm is very sensitive to noise, so most of predictions from OSVM are anomalies. Thus, the OSVM will produce high detection performance when a large portion of data is anomaly as results in Fig. 4. Second, one of the intrinsic features of KNN algorithm is noise tolerance. For this reason, incorrect training data produce a relatively small effect on detection performance as shown in Table 4.

More concretely, the following steps provide guidance to apply our proposed model to real network environments. First, assign a size for the interval value and then consider the trade-offs between the size of interval value and computational complexity as shown in Table 5. However, the interval value will often be assigned by the software of network equipment. Next, select interesting features that you would like to monitor. For example, a specific IP address, a range of IP addresses or even port numbers can be selected as a feature. Lastly, select a learning algorithm by considering the amount of training data, detection performance, and computational time, then set values of related parameters at the midpoint or common value. After operating for a while, administrators could tune these parameters to suit network environment. For example, there are two parameters for the

KNN algorithm, the value of k and the distance value D. A common value of k is 3 and the midpoint of distance D is 0.5. If the detected result shows a high false positive rate, one can increase the value of k or reduce the distance value D. On the other hand for a high false negative rate, one can decrease the value of k or increase the distance value D.

We strongly recommend applying the proposed model to access networks, place it behind a firewall and virus detector, rather than applying to core networks because traffic data on core networks are more diverse than those at access networks. Although we did not examine performance of the proposed model in a core network, the experiments on a core network still needs. However, we firmly believe that the proposed model could be applied to core networks as well.

6. Conclusions

In this article, we have characterized and analysed three fundamental models for anomaly detection in computer networks. We have raised the main problems of each model, and to address these problems, we have proposed an unsupervised learning model for real-time anomaly detection. We have conducted a series of experiments on real network traffic to examine several key aspects of the proposed model, including detection performance, robustness, learning curve, and time consumption. In the experiments, we extracted nine features from network traffic to detect several types of anomalies using both a single feature and a combined feature. In addition, we employed three well-known learning algorithms, namely multivariate normal distribution, k-nearest neighbor, and one-class support vector machine, to work with our proposed model. We measured the detection performance of the proposed model by using F-score which is a standard measurement for binary classifiers.

In summary, the results strongly support the conclusion that our proposed model has the capability to detect anomalies in computer networks with promising performance. The model effectively detects a variety of anomalies caused by attacks or accidents. We also found that the proposed model has flexibility in choosing learning algorithms to work with and choosing features to detect a particular type of anomaly. Furthermore, the model is robust against incorrect training data and data manipulation by attackers. Two of three learning algorithms that worked with the proposed model could quickly learn from training data to detect anomalies. The proposed model not only enables network administrators to detect novel types of attacks but can also be used to identify abnormal behavior of their own networks in real-time.

Acknowledgments

We gratefully acknowledge the funding from the Faculty Members Development Scholarship Program of Bangkok University, Thailand. This research has been partially supported by the Strategic International Collaborative R&D Promotion Project of the Ministry of Internal Affairs and Communication, Japan, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 608533 (NECOMA). We would like to express a special thanks to everyone for their data and support.

References

[1] S. Hansman and R. Hunt, "A taxonomy of network and computer attacks," Comput. Secur., vol.24, no.1, pp.31–43, 2005.
[2] A. Patcha and J.M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," Comput. Netw., vol.51, no.12, pp.3448–3470, 2007.
[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Comput. Surv., vol.41, no.3, pp.15:1–15:58, July 2009.
[4] C.F. Tsai, Y.F. Hsu, C.Y. Lin, and W.Y. Lin, "Intrusion detection by machine learning: A review," Expert Systems with Applications, vol.36, no.10, pp.11994–12000, 2009.
[5] S. Manocha and M.A. Girolami, "An empirical analysis of the probabilistic k-nearest neighbour classifier," Pattern Recognit. Lett., vol.28, no.13, pp.1818–1824, Oct. 2007.
[6] W.H. Chen, S.H. Hsu, and H.P. Shen, "Application of svm and ann for intrusion detection," Comput. Oper. Res., vol.32, no.10, pp.2617–2634, Oct. 2005.
[7] Z. Zhang and H. Shen, "Application of online-training svms for real-time intrusion detection with different considerations," Comput. Commun., vol.28, no.12, pp.1428–1442, 2005.
[8] A. Narasimhamurthy, "Theoretical bounds of majority voting performance for a binary classification problem," IEEE Trans. Pattern Anal. Mach. Intell., vol.27, no.12, pp.1988–1995, Dec. 2005.
[9] M. Roesch, "Snort - lightweight intrusion detection for networks," Proc. 13th USENIX conference on System administration, LISA '99, pp.229–238, Berkeley, CA, USA, 1999.
[10] V. Paxson, "Bro: A system for detecting network intruders in real-time," Comput. Netw., vol.31, no.23-24, pp.2435–2463, 1999.
[11] G. Vigna and R.A. Kemmerer, "Netstat: A network-based intrusion detection system," J. Comput. Secur., vol.7, no.1, pp.37–71, Jan. 1999.
[12] S. Jiang, X. Song, H. Wang, J.J. Han, and Q.H. Li, "A clustering-based method for unsupervised intrusion detections," Pattern Recognit. Lett., vol.27, no.7, pp.802–810, May 2006.
[13] A. Kind, M. Stoecklin, and X. Dimitropoulos, "Histogram-based traffic anomaly detection," IEEE Trans. Network and Service Management, vol.6, no.2, pp.110–121, June 2009.
[14] K. Labib and R. Vemuri, "Nsom: A real-time network-based intrusion detection system using self-organizing maps," tech. rep., University of California, Davis, 2002.
[15] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical real-time intrusion detection using machine learning approaches," Comput. Commun., vol.34, no.18, pp.2227–2235, 2011.
[16] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," Proc. DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00, vol.2, pp.12–26, 2000.
[17] 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. Inf. Syst. Secur., vol.3, no.4, pp.262–294, Nov. 2000.
[18] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z.H. Zhou, M. Steinbach, D.J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," Knowl. Inf. Syst., vol.14, no.1, pp.1–37, Dec. 2007.
[19] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th ed., Academic Press, 2008.

[20] T.M. Mitchell, Machine Learning, 1st ed., McGraw-Hill, New York, NY, USA, 1997.
[21] B. Schölkopf, J.C. Platt, J.C. Shawe-Taylor, A.J. Smola, and R.C. Williamson, "Estimating the support of a high-dimensional distribution," Neural Comput., vol.13, no.7, pp.1443–1471, July 2001.
[22] K.R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," IEEE Trans. Neural Netw., vol.12, no.2, pp.181–201, March 2001.
[23] C.C. Chang and C.J. Lin, "LIBSVM: A library for support vector machines," ACM Trans. Intelligent Systems and Technology, vol.2, pp.27:1–27:27, 2011.
[24] C.J.V. Rijsbergen, Information Retrieval, 2nd ed., Butterworth-Heinemann, Newton, MA, USA, 1979.
[25] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," Proc. 23rd international conference on Machine learning, ICML '06, pp.233–240, New York, NY, USA, 2006.

Kriangkrai Limthong received a B.Eng (2nd Honors) degree in Computer Engineering from Sripatum University; and a M.Eng degree in Computer Engineering from Kasetsart University, Thailand. He worked as a Systems Engineer at Advanced Info Service PLC. and Thailand Post Co., Ltd. for several years. He is currently pursuing the Ph.D. degree in the Department of Informatics, Graduate University of Advanced Studies (Sokendai), Japan. He has also been a lecturer in the Department of Computer Engineering, School of Engineering, Bangkok University, Thailand, since 2009. His research interests are network traffic measurement, computer security, signal processing techniques and machine learning methods.

Kensuke Fukuda is an associate professor at the National Institute of Informatics (NII). He received his Ph.D. degree in computer science from Keio University in 1999. He worked in NTT laboratories from 1999 to 2005, and joined NII in 2006. He was a visiting scholar at Boston University in 2002, and also was a researcher of PRESTO JST (Sakigake) in 2008-2012. His current research interests are Internet traffic measurement and analysis, intelligent network control architectures, and the scientific aspects of networks.

Yusheng Ji received the B.Eng., M.Eng. and D.Eng. degrees in Electrical Engineering from the University of Tokyo in 1984, 1986, and 1989, respectively. She joined the National Center for Science Information Systems in 1990. She is currently an Associate Professor at the National Institute of Informatics and the Graduate University. Her research interests include network architecture, protocols, traffic controls, and performance analysis for quality of service provisioning in wired and wireless networks. She is a member of IEEE, IEICE, and IPSJ.

Shigeki Yamada is a professor and director at the Research Centre for Academic Networks of National Institute of Informatics (NII), Japan. He received his B.E., M.E., and Ph.D. degrees in electronic engineering from Hokkaido University in 1972, 1974, and 1991, respectively. He worked in NTT laboratories from 1974 to 1999, where he was involved in the research and development of high performance digital switching systems and network-wide distributed systems. He moved to NII in 1999. His current research interests include DTN, and mobile/ubiquitous networks as well as network security and privacy technologies. From 1981 to 1982, he was a visiting scientist in the Computer Science Department at the University of California, Los Angeles. He is a senior member of IEEE and a member of IPSJ.