A Queue Model to Detect DDos Attacks

Shuang Hao
haoshuang98@mails.tsinghua.edu.cn

Hua Song
songh00@mails.tsinghua.edu.cn

Wenbao Jiang
jiangwenbao@tsinghua.org.cn

Yiqi Dai
dyq@theory.cs.tsinghua.edu.cn

Department of Computer Science and Technology
Tsinghua University, Beijing 100084, China

ABSTRACT

With the development of network communication and collaboration, distributed denial-of-service (DDos) attack increasingly becomes one of the hardest and most annoying network security problems to address. In this paper, we present a new framework to detect the DDos attacks according to the packet flows of specific protocols. Our aim is to detect the attacks as early as possible and avoid the unnecessary false positive. A Gaussian parametrical mixture model is utilized to estimate the normal behavior and a queue model is adopted for detecting the attacks. Experiments verify that our proposed approach is effective and has reasonable accuracy.

KEYWORDS: Anomaly detection, DDos attacks, Queue model, Gaussian mixture model.

1. INTRODUCTION

In a distributed denial-of-service (DDos) attack, the attacker compromises a number of slaves and installs flooding servers on them, later it contacts the set of servers to combine their transmission power in an orchestrated flooding attack [1]. The typical attacks include smurf, syn flood (neptune) and DNS request flooding, which send a large number of malicious packets to overwhelm the victim's CPU, memory, or network resources. Although the attacks become conspicuous when a system is cracked down or the performance of network service substantially degrades, at that time the victim has already got negative impact. In order to mitigate the damage of DDos, the alerts should be reported before the malicious packets aggregate at the destination. Thus accurate detection is required for prevention and further reaction.

Anomaly Detection paradigm [2] [3] was widely used to address DDos attacks, but there exist two problems which hinder the detecting accuracy.

The first one is that the attributes of DDos attacks are difficult to depict. Some statistical properties of packet stream are deployed to detect flooding behavior. However, it is not certain that the attacks' characteristics can be perceived in the selected observing time scale. Ironically, despite the fact that the packets arrive very quickly in an attack, such situation is also prevalent during a normal connection. Therefore we can not simply declare to have spotted an attack based only on the fast arrivals of packets. Another challenge for anomaly detection is threshold setting. A low threshold leads to many false positives, while a high threshold reduces the sensitivity of the detection mechanism [3]. To establish a suitable threshold is the key problem to solve in the training phase.

Similar to C. Kruegel' work in [4], we built up a model to approximate the distribution of normal observed attributes and detect instances that significantly deviate from the legitimate behavior. Kruegel used the Chebyshev inequality to calculate the upper probability of the deviation. If the distribution of the normal observed attributes has one peak, the detecting results are satisfactory. Otherwise, the method is unable to represent the characteristics on the curve troughs. Compared to his work, we propose a more precise model, Gaussian Mixture Model (GMM), to represent normal behavior of packets' flow in the network. Furthermore, a queue model is developed to evaluate the performance of the observed network. Its service rate is derived from GMM and a reasonable threshold is automatically selected to detect flooding-like behavior.

The rest of the paper is organized as follows. We first introduce our system framework in section 2. In section 3, we construct the data model of the flow behavior. Section 4 demonstrates the mechanisms of attack detecting and threshold determining. Section 5 contains experimental evaluation. Finally we draw conclusion in section 6 with some discussions.

2. SYSTEM FRAMWORK

Our underlying assumption is that a DDos attack injects a huge amount of traffic into the network and this will deviate from the normal behavior. The aim of a detection method is to make the deviation noticeable.

We concentrate on specific protocols respectively, like in [5] [6]. The targets for our analyzing are request and response packets which are vulnerable to existing DDos attacks or possess the potential to suffer new attacks. Some of these specific packets are listed in Table 1 [7].

Table 1. A sample of victim responses to typical attacks

Packet sent	Response from victim
TCP SYN (to open port)	TCP SYN/ACK
TCP SYN (to closed port)	TCP RST (ACK)
TCP ACK	TCP RST (ACK)
TCP DATA	TCP RST (ACK)
TCP RST	no response
TCP NULL	TCP RST (ACK)
ICMP Echo Request	ICMP Echo Reply
ICMP TS Request	ICMP TS Reply
...	...

After receiving a request packet, the response packets are automatically sent or some resources (such as data buffer) will be allocated to expect consecutive communications. Attackers usually abuse these characters to consume the network bandwidth or local host's processing ability.

It is remarkable that the speeds of packets' arrivals are quite diverse and the intervals of successive packets range form millisecond to minute. Thus, the data transferred on the network can be regarded as being issued by generators with different rates. Moreover, the concerned network is treated as a node. Packets of specific protocols flowing in and out the network are as passing customers. The hypothetic node provides a service which will cost some time. The arrived packets waiting for the service are maintained in a queue. The scheduling policy is First Come First Served. Figure 1 shows the abstract framework.

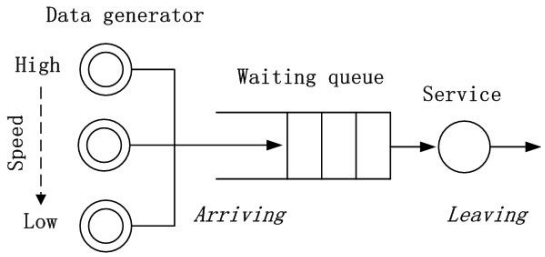


Figure 1. Queue model

The high-speed packets will not gush incessantly in normal situations, so the packets can leak from the queue in our model, which guarantees the queue length (the customer number in the queue) to be small. On the other hand, when the malicious packets with small intervals keep bothering the network, it makes the queue length increase quickly. If the length exceeds a certain threshold, it implies a potential attack.

Several points should be noticed during the construction of the model.

1. The passing directions of the packets through the hypothetic node are not distinguished, for the attacks can occur on either flow directions. In learning and detecting phases, the traffics on both directions are analyzed without discrimination.

2. As we take account of the behavior of the overall flow performance, whether the attacker spoofs the source addresses does not influence the accumulation of the malicious packets in the queue and the attack will be spotted.

3. Based on the characteristics of DDos attacks, it is hard to bypass our detecting mechanism. Otherwise attackers have to slow down the rate of sending packets, which mitigates the damage of attacks.

4. In a DDos attack, the malicious flows usually originate from different networks simultaneously. Because the traffic outgoing from the network is under the inspecting scope as well, it is possible that the attack is detected and stopped within the stained networks and the victim network is saved before severe congestion occurs.

3. BEHAVIOR MODEL

The task of modeling is to represent the characteristics of different speed generators in Figure 1. We model the

interval between adjacent passing packets as a continuous random variable x. As the intervals usually congregate on enormously different time scales, e.g. from 0.001s to 100s, it seems flimsy to directly set up a model for describing the distribution of x. We make a translation, $y = \ln(x)$, $x > 0$, and take y as the concerned variable. There exists little priori knowledge about the reference data, because the topology of the network and the concerned protocols are varied. Generally, we suppose that y is able to be represented by a Gaussian Mixture Model as stated in Equation (1).

$$p(y) = \sum_{k=1}^{K} \omega_k \phi(y, \mu_k, \sigma_k^2) \tag{1}$$

Where $p(y)$ denotes the probability density function; K is the kernel number of the model; ϕ_k represents each Gaussian kernel function with mean μ_k and variance σ_k^2; ω_k is the pondering factor that indicates which source data comes from. In this paper, we just discuss the unidimensional case. Then the data created by different resource generators (like in Figure 1) are approximated by different Gaussian kernel distributions.

$\Theta_K = (\omega_1, \ldots, \omega_k; \mu_1, \ldots, \mu_k; \sigma_1^2, \ldots, \sigma_k^2)$ represent the unknown parameters in Equation(1), which are managed to calculate during the training phase. Unsupervised learning is accomplished by the EM algorithm [8]. The expectation and maximization steps are simply stated as in Equation(2)∼(5).

$$p(k|y_i, \Theta_K^g) = \frac{\omega_k^g \phi(y_i, \mu_k^g, \sigma_k^{2g})}{\sum_{m=1}^{K} \omega_m^g \phi(y_i, \mu_m^g, \sigma_m^{2g})} \tag{2}$$

$$\omega_k^{new} = \frac{1}{N} \sum_{i=1}^{N} p(k|y_i, \Theta_K^g) \tag{3}$$

$$\mu_k^{new} = \frac{\sum_{i=1}^{N} y_i \times p(k|y_i, \Theta_K^g)}{\sum_{i=1}^{N} p(k|y_i, \Theta_K^g)} \tag{4}$$

$$\sigma_k^{2new} = \frac{\sum_{i=1}^{N} (y_i - \mu_k^{new})^2 \times p(k|y_i, \Theta_K^g)}{\sum_{i=1}^{N} p(k|y_i, \Theta_K^g)} \tag{5}$$

Where superscript g indicates the guessed appropriate parameters in the last iteration when K kernels are assumed; N is the number of training data; y_i denotes the i-th observed datum. As EM algorithm has a well-known failure mode to converge to a local maximum of the log-likelihood result, we deploy multiple initial parameters to fit the model and choose the maximum likelihood as the optimized model parameter Θ_K^o under K kernels.

$$\Theta_K^o = \arg\max_{\Theta_K} \left(\sum_{i=1}^{N} \ln(p(y_i|\Theta_K)) \right) \tag{6}$$

Even if the optimized model parameters are obtained, the kernel functions' number K is still undetermined. As presented in [9], an optimal entropy-based approach is used to build the "ideal partitioning", which is achieved by having, for each datum, some partition posterior $p(k|y)$ close to one, while all the others are close to zero. Furthermore, to optimize the partition is equivalent to minimize the Shannon entropy given observed datum y in Equation(7).

$$H(y) = -\sum_{k=1}^{K} p(k|y) \log(p(k|y)) \tag{7}$$

To take account of all observed data Y, the entropy change is given as

$$\triangle H(\Theta_K|Y) = H(\Theta_K) - H(\Theta_K|Y) \tag{8}$$

where $H(\Theta_K)$ indicates the model entropy prior to the observed data and each partition probability is equal to $1/K$; $H(\Theta_K|Y)$ denotes the entropy with posterior partition probabilities after having observed Y.

The maximum value of entropy change is regarded as the most appropriate candidate for the model partition, as shown in Equation(9).

$$K_{opt} = \arg\max_{K} (\triangle H(\Theta_K|Y)) \tag{9}$$

Based on the above description, we successfully construct a Gaussian Mixture Model, in which each kernel generator is inclined to create a cluster of observed data and the overlapping influence of different kernels to produce a reference datum is as little as possible.

4. DETECTING MECHANISM

Our objective is to optimize the service time in the queue model, which should not be extremely long to arouse overwhelming false positive or extremely short to cause excessive false negative.

As the behavior model is constructed by different-rate generators, we arrange the kernel functions in ascending order by their mean values. Then the probability density function of y (which stands for the logarithm of intervals of packets) is re-expressed as below.

$$p(y) = \sum_{k=1}^{K} \omega_{(k)} \phi(y, \mu_{(k)}, \sigma_{(k)}^2) \tag{10}$$

$$\forall\, 1 \le i < j \le K, \mu_{(i)} < \mu_{(j)}$$

The kernel generator with higher rate has smaller subscript. In the rest of the paper, we will use the subscript

numbers in Equation(10) to denote each data generators.

The highest-rate generator is identified as f and the second one is denoted as m. The service time is assumed larger than the arriving interval of kernel f but smaller than that of kernel m, which ensures the 'fastest' packets are detained in the queue.

4.1. Filter The Noise

The kernel weights are examined by the sequence in Equation(10) and the extremely small ones are ignored, which may indicate the noises brought up from data modeling phase. So the appropriate candidate f is determined as the minimum number to satisfy the inequality shown below to guarantee that the expected weight is larger than a ratio of $1/K$.

$$\sum_{k=1}^{f} \omega_{(k)} > \lambda \frac{f}{K} \tag{11}$$

Where λ is the ratio value configured by operators. We set $\lambda = 1/3$ in the following experiment. f is equal to 1 if there is no noise. Note that $f = K$ means either λ is inappropriate or the model fails.

Similarly, the second fastest generator is m, the minimum number to satisfy Inequality(12).

$$\sum_{k=f+1}^{m} \omega_{(k)} > \lambda \frac{m-f}{K} \tag{12}$$

We note that $m = f + 1$ if no noise exists between the two generators.

4.2. Locate The Boundaries of Kernels

The range of data created by a generator is limited by the relevant kernel's upper and lower boundaries. The Gaussian kernel boundary is defined as the point where probability density function (*pdf*) first drops to a low threshold (nearly zero). If the distance of a point away from mean μ is d, the ratio of its function value to the acme of the curve is

$$R(d) = \exp(-\frac{d^2}{2\sigma^2}) \tag{13}$$

It is easy to know that
$R(\sigma) = 0.6065$, $R(2\sigma) = 0.1353$, $R(3\sigma) = 0.0111$, $R(4\sigma) = 0.0003$. We regard 3σ as a satisfactory distance to meet the requirement of the kernel boundary. The lower and upper boundaries of a kernel are shown in Equation(14)(15).

$$B^l = \mu - 3\sigma \tag{14}$$

$$B^u = \mu + 3\sigma \tag{15}$$

The logarithm of service time, $\ln(T_s)$, will be equal to one of these two values, the upper boundary of kernel f or the lower boundary of kernel m.

4.3. Decide The Final Service Time

As shown in Figure 2, if the *pdf* of the two components f and m are well separated, the logarithm of service time should be around the mark B_f^u where *pdf* of kernel f drops to nearly zero.

$$T_s = \exp(B_f^u) \quad \text{if} \quad B_f^u \leq B_m^l \tag{16}$$

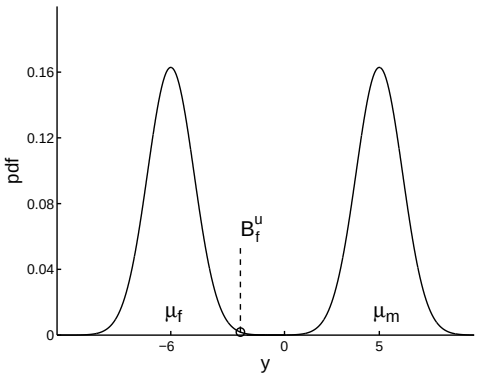
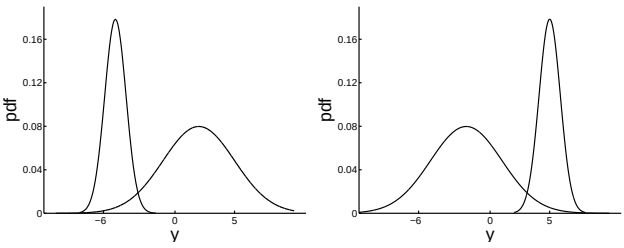


Figure 2. *pdf* of components well separated ($B_f^u \leq B_m^l$)



(a) $\sigma_f \leq \sigma_m$ (b) $\sigma_f > \sigma_m$

Figure 3. *pdf* of components overlapped ($B_f^u > B_m^l$)

If the two components overlap as shown in Figure 3, constructing a generator with smaller standard deviation empirically leads to higher reliability and the service time is calculated as follows.

$$T_s = \begin{cases} \exp(B_f^u) & \sigma_f \leq \sigma_m \\ \exp(B_m^l) & \sigma_f > \sigma_m \end{cases} \quad \text{if} \quad B_f^u > B_m^l \tag{17}$$

So, the basic service time, T_s, is attained from Equation(16)(17).

Although, under the service time T_s, the system is efficient to detect attacks, but it will hold enormous packets' information in the queue after a flooding attack. It is confusing to affirm when the attack is over. Furthermore, even after the attacks the queue length will exceed the threshold and the system will keep reporting false alarms. In order to solve this problem, the service time is set shorter when the queue length exceeds the threshold, as shown in Equation(18). It makes the queue length shrink quickly after the attacks and reduces the false positive dramatically.

$$t_s = \begin{cases} T_s & l \leq l_{threshold} \\ \frac{l_{threshold}}{l} T_s & l > l_{threshold} \end{cases} \quad (18)$$

Where t_s is the dynamic service time during the detection; l indicates the queue length; $l_{threshold}$ is the threshold of the system.

4.4. Detecting Procedure

Queue length l denotes the number of packets in the queue, set $l = 0$ at the beginning; The dynamic service time t_s is achieved from above steps; $l_{threshold}$ stands for threshold of the queue length to raise an alarm. Its determination will be explained latter; The server S has two states, b(busy)/ i(idle), and is set to be in state i initially.

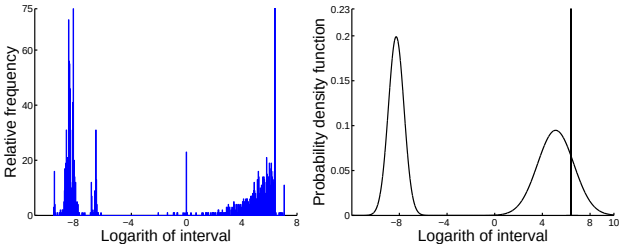
The detecting procedure is summarized as follows,

1. When the specific packet passes the network, its information is stored in the queue. Let $l = l + 1$.
2. Whenever $S = i$ and $l \neq 0$, a packet in the queue is forwarded to receive the hypothetic service. Let $S = b$. After t_s, the service will be finished.
3. No sooner does one service end than the information of the served packet is deleted and let $l = l - 1$, $S = i$.
4. During above steps, if $l > l_{threshold}$, the detecting alarm is positive; otherwise, it is negative.

As the service time is elaborately selected, we impose the training data (without attacks) on the system again when maintaining $t_s = T_s$, and assume the maximum queue length is the detecting threshold. This configuration makes the model tolerant to slight deviation of legitimate packets but sensitive to the malicious ones.

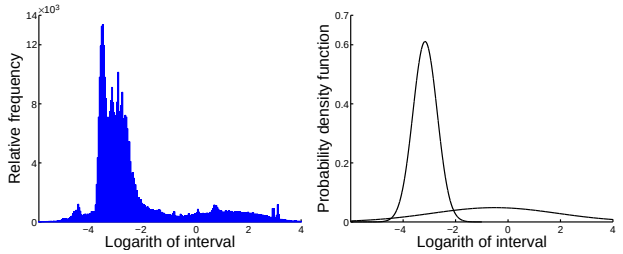
5. EVALUATION

We used the MIT Lincoln Labs 1999 data set [10] [11] to evaluate our approach. Lincoln Labs set up an environment to acquire raw packet dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. The attack-free data of two weeks are used to train the data model and we ran the detection system on the test data that were recorded during another two weeks. Although several attributes of the Lincoln Labs data set were criticized and the evaluating results have the possibility to be overoptimistic [12], it is the most widely used public benchmark for testing intrusion detection system.



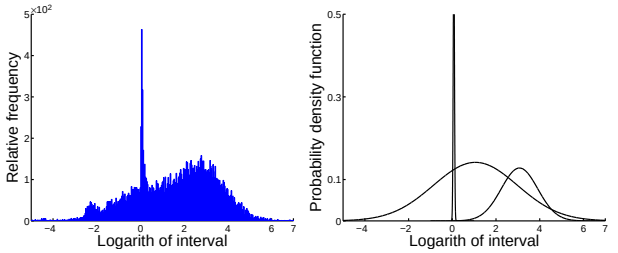
(a) Histogram (b) Distribution

Figure 4. Modeling of logarithm of icmp-reply packets' arriving interval (inside sniffer)



(a) Histogram (b) Distribution

Figure 5. Modeling of logarithm of tcp-syn packets' arriving interval (outside sniffer)



(a) Histogram (b) Distribution

Figure 6. Modeling of logarithm of tcp-syn (to port 25) packets' arriving interval(outside sniffer)

Our concerned classes of Dos attacks are those that abuse a legitimate feature and issued enormous packets to the victim during a short time. The MIT Lincoln Labs data set contains several instances of such attacks: smurf, syn flood (neptune) and mailbomb, which abuse the features of

icmp-reply packets, tcp-syn packets and tcp-syn packets to port 25 respectively.

From Figure 4 to Figure 6, we show the histogram of logarithm of packets' arriving intervals in the attack-free data and the resultant models by using EM algorithm and entropy partition.

The data models are highly accordant with the realistic data. Week 4 and week 5 dumped data are used to test the efficiency of our approach. After running the detecting process, the results are shown in Table 2. The 'Kernel's boundary' item indicates which kernel in the models determines the final service time, e.g. '(1)-upper' means the logarithm service time is derived from the upper boundary of the first kernel.

Table 2. Testing results

Result / Attack	True positive	False positive	Kernel's boundary
smurf	99.86%	0.000%	(1)-upper
neptune	99.08%	0.0348%	(1)-upper
mailbomb	90.82%	0.908%	(3)-lower

Our approach maintains a low false positive and a satisfactory detecting rate. In most cases, the number of false alarms per day is less than 50. The testing results revealed the inclination that the better the kernels in the model are separated, the more accurate result is expected.

To be more specific, we focus on the case of smurf attack below. The threshold about icmp-reply packets after training is equal to 9, which indicates legitimate behavior will not cause an excessive length. The length of the queue during the detection is illustrated in Figure 7. For displaying convenience, we only show the days when the queue length appears abnormal. Note that in other days the length never exceeds the threshold.

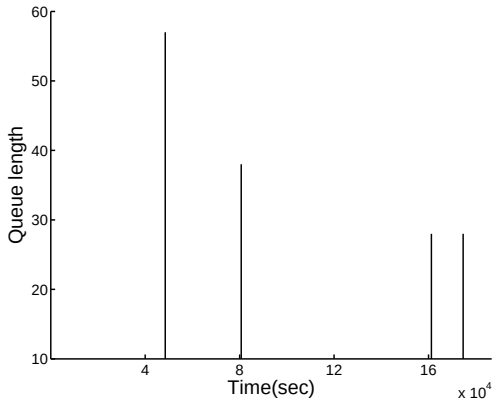


Figure 7. Queue length during testing

There are totally 4 extreme bursts during the test. We list the corresponding attacks by the time sequence in Table 3.

Table 3. Smurf attacks related with icmp-reply packets (item 'Day' indicates week-day of the attack. 'Victim IP' is always 172.16.xx.xx, so we just show the last two bytes in the table)

Day	Start Time	Duration (sec)	Packets Number	Victim IP
4-1	21:34:05	15.3549	51680	112.50
4-5	08:45:14	2.0259	4452	112.50
5-1	09:34:10	3.9602	6000	112.50
5-1	13:18:06	1.6522	2653	114.50

According to the above presentation, the estimated service time is obviously a crucial factor to affect the detecting efficiency. To make a comparison, we take two other settings for the service time, one uses the mean of intervals and the other uses the exponent of fastest kernel's mean ($\exp(\mu_f)$). The results about icmp-reply packets are listed in Table 4.

Table 4. Comparison (detecting smurf) between different service time defined by (1) our approach (2) mean of intervals (3) $\exp(\mu_f)$

	(1)	(2)	(3)
Basic service time (s)	0.00182	295.8	0.000257
Max length at training	9	115	3
Max length at testing	56	51686	3
True positive (%)	99.86	99.52	0.0
False positive (%)	0.0	22.38	0.0

As demonstrated in Table 4, if we just consider the mean of the intervals, information about many normal packets is stored in the queue as well and the false positive will be raised. That may hamper the further reaction after detecting attacks, e.g. finding out the attacker's IP address. On the other hand, if the service time is too small, the queue length will almost not vary in training and detecting. Then the model will totally fail. It proves that our method has a tradeoff between the large and small service time, and performs better in most cases.

6. CONCLUSION

This paper has presented a new anomaly detection method to spot DDos attacks. Our contribution is an efficient framework that depicts detailed characteristics of

packet flows and makes full use of them in the detecting phase. The queue model and the threshold are easy to implement and need few human interactions. The experimental results show that the assumption corresponds with the realistic situation and the system owns high sensitivity to malicious packets but low false positive to legitimate ones.

As an approach of anomaly detection, our framework has a drawback: if the network environment changes enormously after training, the threshold and the hypothetic service time might jeopardize to fool the detecting results. Awareness of the failure and update of the model is the focus of future work. To compensate for the defect, combining our approach with other detecting and preventing mechanisms can be recommended.

References

[1] V. Paxson, "An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks", *ACM SIGCOMM Computer Communication Review*, Vol.31, No.3, 2001, pp. 38-47.

[2] D. E. Denning, "An Intrusion-Detection Model", *IEEE Transaction on Software Engineering*, Vol.13, No.2, 1987, pp. 222-232.

[3] J. Mirkovic, P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms", *ACM SIGCOMM Computer Communication Review*, Vol.34, No.2, 2004, pp. 39-53.

[4] C. Kruegel, G. Vigna, "Anomaly Detection of Web-based Attacks", Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington D.C., October 2003.

[5] R. Sekar, A. Gupta, J. Frullo, "Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions", Proceeding of the 9th ACM Conference on Computer and Communications Security, Washington D.C., November 2002.

[6] M. Mahoney, "Network Traffic Anomaly Detection Based on Packet Bytes", Proceedings of the 2003 ACM Symposium on Applied Computing, Melbourne, March 2003.

[7] D. Moore, G. M. Voelker, S. Savag, "Inferring Internet Denial-of-Service Activity", Proceedings of the 10th USENIX Security Symposium, Washington D.C., August 2001.

[8] J. A. Bilmes, "A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models", Technical report, icsi-tr-97-021, UC Berkeley, April 1998.

[9] S. J. Roberts, R. Everson, I. Rezek, "Maximum Certainty Data Partitioning", *Pattern Recognition*, Vol.33, No.5, 2000, pp. 833-839.

[10] MIT Lincoln Labs, DAPAR Intrusion Detection Evaluation, http://www.ll.mit.edu/IST/ideval/, 1999.

[11] R. Lippmann, J. W. Haines, D. Fried., "The 1999 DARPA Off-Line Intrusion Detection Evaluation", *Computer Networks*, Vol.34, No.4, 2000, pp. 579-595.

[12] M. Mahoney, P. K. Chan, " An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation", Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection, LNCS 2820, Pittsburgh, September 2003.