Hierarchical Kohonenen Net for Anomaly Detection in Network Security

Suseela T. Sarasamma, Qiuming A. Zhu, and Julie Huff

***Abstract*—A novel multilevel hierarchical Kohonen Net (K-Map) for an intrusion detection system is presented. Each level of the hierarchical map is modeled as a simple winner-take-all K-Map. One significant advantage of this multilevel hierarchical K-Map is its computational efficiency. Unlike other statistical anomaly detection methods such as nearest neighbor approach, K-means clustering or probabilistic analysis that employ distance computation in the feature space to identify the outliers, our approach does not involve costly point-to-point computation in organizing the data into clusters. Another advantage is the reduced network size. We use the classification capability of the K-Map on selected dimensions of data set in detecting anomalies. Randomly selected subsets that contain both attacks and normal records from the KDD Cup 1999 benchmark data are used to train the hierarchical net. We use a confidence measure to label the clusters. Then we use the test set from the same KDD Cup 1999 benchmark to test the hierarchical net. We show that a hierarchical K-Map in which each layer operates on a small subset of the feature space is superior to a single-layer K-Map operating on the whole feature space in detecting a variety of attacks in terms of detection rate as well as false positive rate.**

***Index Terms*—Computer network security, neural network applications, self-organizing feature maps.**

I. Introduction

RELIABLE network communication is crucial in the day-to-day functioning of the modern world. Businesses, educational institutions, government departments, and even the average individual rely heavily on the uninterrupted communication network service and computing facilities to conduct the day-to-day operations. As the information super highway has become more and more critical to our daily lives, attempts to disrupt the dependable network flow have also increased significantly. Network-based computer systems have increasingly become the target for attackers whether they are just hackers, enemies, or criminals. Many of these attacks are costly in terms of financial losses.

A significant amount of research has been done in the area of detecting intrusions in computer systems over the past 20 years. From an architectural perspective, an intrusion detection system (*IDS*) can be one of three basic types [20]: network-based IDS, host-based IDS, or hybrid IDS. Network-based IDS monitor the IP packets, usually packet headers flowing over the network. Host-based IDS use the audit data of host machines. Hybrid IDS apply both the above methods to capture intrusions from outside, as well as from within. Algorithmically, there are two different approaches commonly used in detecting intrusions [12]. The first approach, commonly known as misuse detection, is a rule-based approach that uses stored signatures of known intrusion instances to detect an attack. This approach is highly successful in detecting occurrences of previously known attacks. However it fails to detect new attack types and variants of known attacks whose signatures are not stored. When new attacks occur, the signature database has to be manually modified for future use. The second approach is commonly known as the anomaly detection approach. In this approach, usually a profile for normal behavior is first established. Then deviants from the normal profile are considered as anomalies. In some cases, these anomalies may be just normal operations that are exhibiting some behaviors adherent to unseen mode of operation. In such cases, the anomalies may be showing false positives. That is, classifying a normal behavior as abnormal, and hence as possible attack instances. One of the techniques used for anomaly detection is building statistical models using metrics derived from observation of the user's actions [12], [21], [25], [26]. One of the metrics used is an event counter that represents the number of events in a set time interval, for example, the number of connections made to different destinations from the same source IP address with in an hour. Another metric that is commonly used is the time interval between two correlated events. A third metric that is generally used is the resource usage such as CPU time consumption, number of files opened, etc. A complete listing of the statistical models built on these metrics can be found in [12].

Anomaly detection schemes also use data mining techniques such as clustering, support vector machines (SVM), and different neural network models [11], [16], [17]. Some of these anomaly detection techniques are capable of detecting new kind of attacks since the techniques are mostly data-driven and do not depend on previously observed patterns and stored signatures. However, these techniques often result in high false positive rates [15]. For example, Sung *et al.* applied a SVM technique to realize an intrusion detection system for class-specific detection [27]. They applied a "leave-one-out" approach to discriminate features for their individual significance, and relied on one SVM for detecting each attack type. However, the approach is intrinsically faulted on totally ignoring the coherent relations and dependencies of the features. Maxion and Tan conducted some experiments to prove that differences in data regularities

Manuscript received May 3, 2004; revised August 25, 2004. This work was supported in part by the Air Force Research Laboratory, Rome, New York and by the Advanced Research Development Agency under Grant F30602-03-C-0247. This paper was recommended by Associate Editor W. Y. Wang.

S. T. Sarasamma and J. Huff are with the Northrop Grumman Mission Systems, Bellevue, NE 68005 USA (e-mail: suseela_ts@ yahoo.com; julie.huff@ngc.com).

Q. A. Zhu is with the College of Information Science and Technology, University of Nebraska at Omaha, Omaha, NE 68182 USA (e-mail: qzhu@mail.unomaha.edu).

Digital Object Identifier 10.1109/TSMCB.2005.843274

can influence anomaly detector performance such as increasing the false positive rate [15]. In their study, they hypothesize that deploying a particular anomaly detector across several environments may not guarantee uniform performance due to differences in data regularities.

In this paper, we present a multilayer hierarchical Kohonen Net, or Kohonen self-organizing map (K-Map) [18], to implement an anomaly based intrusion detection system (IDS sensor). We did our training and testing using the pre-processed KDD Cup 1999-benchmark data [1], [2]. Our objective was to detect as many different types of attacks as possible. The experimentations were conducted in two steps. First, we used a single-layer winner-takes-all K-Map to do an implementation of IDS. We will soon see that there are serious limitations for a single-layer winner-takes-all K-Map in detecting a vast set of attacks and still keeping a low false positive rate [19]. The single-layer winner-takes-all K-Map is a useful technique generally in the sense that it helps to group similar input vectors into clusters [23], [24]. However, it does not guarantee optimal separation of resulting clusters—see Fig. 1(a) and (b), for an example. Here, **x** and **o** indicate two different types of objects, where objects of type **x** are of similar kind and objects of type **o** are of similar kind. Fig. 1(a) shows the cluster 1 and cluster 2 that resulted when a two-dimensional subspace mapping was used and Fig. 1(b) illustrates the grouping when a one-dimensional (1-D) subspace mapping was used for the classification. We can see that neither cluster 1 nor cluster 2 is homogeneous in Fig. 1(a). However the clusters 1 and 2 in Fig. 1(b), obtained using the 1-D subspace mapping, are homogeneous (The example does not imply that a subspace mapping is always better than a whole-space mapping. The case should not be generalized.). That is, the clusters can vary significantly depending on the dimensionality of the clustering boundaries. This observation forms the conceptual basis that leads us to the development of a multilayer hierarchical K-Map in our second step experimentation for an anomaly detection based IDS.

In the context of IDS for network traffic, we observe that there are specific features in the packet headers that are more significant indications of abnormal activities [5]. The input vectors used for grouping packets into normal or attacks of a specific type will be made up of such features. For instance, the flag along with the number of wrong fragments and the service used may indicate a certain type of anomaly. On the other hand, some specific features alone will not be a good indicator of any kind of anomaly.

For instance, the number of bytes that were transmitted from the destination to the source cannot by itself be an indicator of anomalous behavior since it can have a wide range for normal activities alone. However, when in conjunction with other features such as the protocol used, the number of packets from source to destination, the number of wrong fragments, etc., they together will be a better indicator of anomalous behavior. It is because a network attack may not happen as one single action. It may be preceded by many small seemingly innocuous actions. Prior to attacking one might do several small probe actions that are temporally spaced. Or they could launch a massive denial of service attack. Each variation of attack will manifest with a certain pattern of a particular set of features. So it seems more

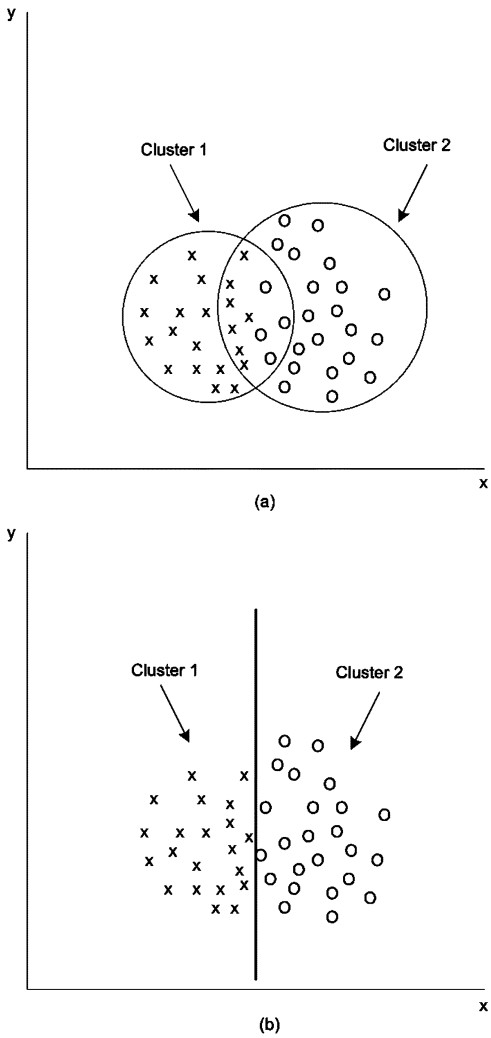

Fig. 1. Effect of feature space dimensions on clustering.

beneficial (efficient) to apply a multilayer hierarchical approach for the K-Map with a set of features carefully selected for each clustering level.

However, one issue comes up when we try to do this multilevel detection. That is, what set of features should be applied at each detection level of the multilayer K-Map. To tackle this problem, we applied a confidence metric as a measure of the mapping quality on the K-Map outputs. We compute a confidence factor for each neuron that will indicate the level of confidence or certainty with which we can say a cluster mapped to a neuron is normal or of a specific type of anomaly. If a group of input feature vectors are being grouped to a neuron's cluster in level i with 100% certainty, then we consider that set *reliably classified* at level i. Those input feature vectors that were mapped to reliably classified clusters will not need to be fed to subsequent levels for further processing.

In this paper, we first present our experiments with a single-layer winner-takes-all K-Map and then with a multilevel hierarchical winner-takes-all K-Map in detecting anomalies in the benchmark KDD Cup 1999 data set. MIT Lincoln Laboratory created the data sets for their IDS evaluation contests conducted in 1998 and 1999. Detailed description of the 1998 DARPA off-line intrusion detection contest can be found in [13]. Some of

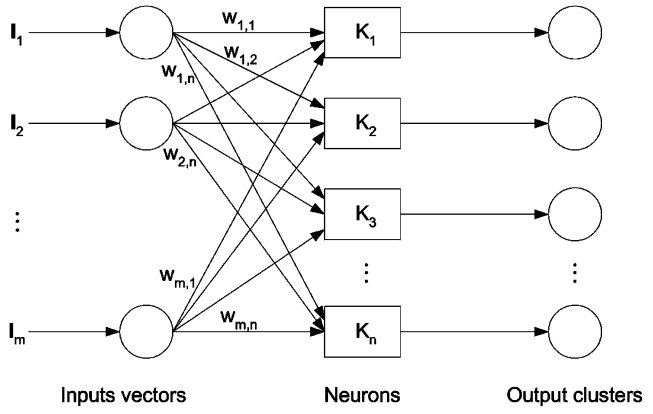

Fig. 2. Single-layer winner-takes-all Kohonen map.

the shortcomings of the 1998 and 1999 DARPA intrusion detection evaluation are identified in [14]. This data set has served as the first and only reliable benchmark data set that has been used for most of the research work on intrusion detection algorithms. Stolfo *et al.* extracted basic features and derived secondary features [2] from these original data and created the KDD Cup 1999 data that was used for the Third International Knowledge Discovery and Data Mining Tools competition held in conjunction with KDD-99. This data set is archived at the University of California, Irvine site [3]. This derived set has also been commonly used as a benchmark for evaluating anomaly detection algorithms. The training set and the test set are organized in files as connection records in ASCII format, where each record contains the coma-delimited set of 41 features and the label indicating whether the record is normal or an attack. For our current experiments, we use the derived data set.

The rest of this paper is organized as follows. In Section II, we first present a brief description of a single-layer winner-takes-all K-Map. Then we present the training algorithm that we used for the simple K-Map. Section II-A discusses some implementation aspects for the K-Map. Section II-B discusses the selection of feature vectors and more details on training and testing. The preliminary results for the simple K-Map are also presented in Section II. Section III describes a three-level hierarchical winner-takes-all K-Map, as an implementation of our multilayer K-Map for IDS. We describe the training, testing, and observed results of the multilayer hierarchical K-Map in Section III. Section IV describes related work done by other researchers, and concludes our presentation.

II. Anomaly Detection Using Single-Layer Winner-Takes-All Kohonen Map

A. Single-Layer Winner-Take-All Kohonen Map

A simple winner-takes-all Kohonen Net consists of an input layer, a neurons layer and an output layer [4], [18]. Since it only employs one neuron layer, we thus also call it a single-layer K-Map. The neurons layer consists of a set of neurons that can be visualized as arranged in a column—see Fig. 2 for a graphical illustration. Each neuron has an associated weight vector. The input layer serves to feed each neuron from a set of input vectors to the different neurons in the neurons layer. An input vector is tied to a neuron by its associated weight vector. The output layer represents a set of clusters, one for each neuron. The elements of an output cluster, say out_i, represent the group of input vectors that are closest to the weight of neuron i. The best matching neuron for an input vector is determined based on the dot product of the normalized input vector and the normalized weight vector of that neuron. This value is commonly known as the net_i. Each input vector is fed to all the neurons. The neuron that gives the maximum value for net_i is declared the winner, or the best matching unit. In other words, only one neuron fires for a specific input vector. Note that the definition of winner-takes-all map in the context of this paper means that only one neuron fires at the output layer. This makes the algorithm a low-cost implementation of self-organizing map.

Like any other neural network, the K-Map needs to be trained. The training is achieved by an unsupervised learning algorithm, which is described in the pseudo code, *Algorithm train_simple_winner_take_all* below. For better readability of the pseudocode, we give a brief description of the symbols used. In the context of the pseudocode, α represents the learning factor for the current iteration. Current iteration is indicated by the variable $\mathrm{loopCount}$. The initial learning factor is denoted as α_0. The dot product of an input vector for the ith neuron is denoted as net_i.

```
Algorithm Train_Simple_Winner_Take_All
Step 1: Obtain the following inputs from
        the user.
          • Number of neurons, numNeurons
          • The size of the input vector,
            numFeatures, that is determined from
            the feature subset to use
          • The threshold to determine the number
            of training iterations
          • The names of input and output files,
            where input file represents the file
            in which the training data set is
            located and output file represents
            the file to which the trained state
            of the K-Map is stored.
Step 2: Read input records.
Step 3: Construct input vectors with the
        specified subset of features. Let
        numVectors denote the total number of
        vectors.
Step 4: Construct a single-layer K-Map
        with the specified number of
        neurons (numNeurons) and the selected
        number of features (numFeatures).
        The K-Map will have a weight matrix of
        size numNeurons × numFeatures.
Step 5: Initialize the weight matrix with
        randomly selected input vectors and
        normalize the weight matrix.
Step 6: Initialize loopCount : loopCount ← 0;
        Choose an initial learning factor
        α0 ← 1.
Step 7: Compute maxChanges ← numVectors ×
        threshold
```

Step 8: **Repeat**

loopCount ←loopCount+1

$\alpha \leftarrow \alpha_0/\sqrt{\text{loopCount}}$

numChanged ← 0

For each input vector I do

For each row of the weight matrix, compute net_i

$$net_i \leftarrow \sum_{k=0}^{\text{numFeatures}-1} W_{i,k} * I_k$$

Choose the winner as the neuron j, where

$$net_j = \max(net_i), \quad 0 \le i < \text{numNeurons}$$

Adjust the weights for the winner as

$$W_{\text{winner},k} \leftarrow W_{\text{winner},k} + \alpha * (I_k - W_{\text{winner},k}), \quad 0 \le k < \text{numNeurons}$$

Normalize the weight for the winner. If the existing best matching unit for I, map $(I) \neq$winner

$$\text{numChanged} \leftarrow \text{numChanged} + 1$$

until numChanged < maxChanges

B. Implementation of a Single-Layer K-Map in Network Intrusion Detection

One aspect that makes a Network Intrusion Detection System (NIDS) successful is its protocol awareness. An anomaly detection scheme is just one piece of the puzzle in the NIDS. In our experiments, we studied the effect of different protocol-specific aspects of the packet headers in the performance of the single-layer K-Map for anomaly detection. In the context of the KDD Cup 1999 data set, each record encapsulates the basic traffic features, the features that are derived by observing the past 100 connections, the features observed in the past two seconds, and some content features. We trained the single-layer K-Map with selected subsets of the KDD Cup 1999 record's features and stored the state of the trained K-Map for each such subset in specific files. The test data sets were run against each such feature subsets at a later time by pre-loading the stored state from the corresponding state file. It is desirable to see the effect of the number of neurons used, and the number of training iterations in the performance of the K-Map. With these in mind, a graphical user interface was designed that drove the training and testing of the K-Map using various parameters.

Some anomaly detection algorithms for NIDS use attack-free or normal data for training. Then test data that contains attacks are fed to the system to detect anomalies. In our tests we first conducted experiments using training data that contains both attacks and normal data. We then examined the composition of the resulting clusters formed at each neuron. Some of these clusters contain records of a specific label, say normal only. We call such clusters homogeneous clusters. Others may contain records of two or more different types, such as "smurf" and "normal" types. We call such clusters heterogeneous clusters. We label the homogeneous clusters with the type of its member records. As for heterogeneous clusters, we use the following technique:

1) Let X_i denote the number of records of type X in cluster i, and X_{total} the total number of records with label X in the training set. Then the probability of a record of type X mapping to cluster i is taken as X_i/X_{total}.
2) Let N_i denote the total number of records in cluster i. The probability of a record mapped to cluster i being of type X is taken as X_i/N_i.
3) We define a favorable factor for label X as the joint probability of a record of type X from the set of inputs mapping to cluster i and a record that is mapped to i being of type X.

$$F_i(X) = \left(\frac{X_i}{X_{\text{total}}}\right)\left(\frac{X_i}{N_i}\right).$$

Let $A = \{a_1, a_2, \ldots, a_r\}$ denote the set of labels identifying all the records that mapped to neuron i. We define label of i as $L(i) = a$, where $F_i(a) = \max_{x \in A} F_i(x)$. We also define a *confidence factor* that is the level of confidence with which we choose the type of record that dominates the cluster. The confidence factor serves as a base for the multilevel K-Map construction. It is used to choose the data set for training and the feature subspace to be involved in the next level of training. The confidence factor of cluster is computed as

$$C_i = \frac{\max F_i(x)}{\sum\limits_{x \in A} F_i(x)}.$$

We define a false positive as an instance where a normal record is identified as an anomaly. If an instance of anomaly is identified as normal, it is a case of missed detection. Let N_{false} denote the total number of false positives encountered in the test, N_{normal} the total number of normal records in the test set, N_{attack} the total number of attacks in the test set and N_{missed} the total number of missed instances. The percentage of false positive is computed as $\%\text{FalsePositive} = (N_{\text{false}}/N_{\text{normal}}) * 100$. The percentage of detected anomalies is computed as $\%\text{Detected} = (N_{\text{attack}} - N_{\text{missed}})/N_{\text{attack}}{}^{*}100$. Note that the simple K-Map could mistake one type of attack as another type of attack. For instance, a snmpguess type attack may be identified as a smurf attack. However at this time we are only interested in knowing whether the record is normal or anomalous.

The KDD Cup 1999 benchmark training data set we used contains 494 021 records (Table I), each with 41 features. There are 22 different attack types in addition to normal records. We created several subsets of these 494 021 records as training sets. We list the composition of four of these training sets identified as Set1, Set2, Set3 and Set4 in Table I. Set1 is selected such that it has records of all 22 attack types with a total of 43 752 records. There are 44 000 records in both Set2 and Set3, each with 16 and 15 attack types respectively. These records are selected randomly from the 10% KDD Cup 1999 training set such that there are no duplicates in a single set. Different seeds are used for the random number generator for each of the three training sets.

TABLE I
COMPOSITION OF TRAINING AND TESTING DATA SETS

Type	KDD99 10%	Set 1	Set 2	Set 3	Set 4	Set 5	Test set1	Test set2
normal	97,278	25,000	8,600	8,740	97,278	20,676	60,593	60,593
neptune	107,201	5,000	25,000	9,491	45,700	22,783	58,001	58,001
smurf	280,790	5,000	9,610	24,971	17,270	0	164,091	0
back	2,203	2,203	201	192	2,203	0	1,098	0
satan	1,589	1,589	145	148	1,589	316	1,633	1,633
ipsweep	1,247	1,247	116	101	1,247	0	306	0
portsweep	1,040	1,040	93	106	1,040	225	354	354
warezclient	1,020	1,020	84	86	1,020		0	
teardrop	979	979	97	97	979		12	
pod	264	264	23	28	264		87	
nmap	231	231	16	29	231		84	
Guess_passwd	53	53	6	5	53		4,367	
Buffer_overflow	30	30	2	1	30		22	
Land	21	21	1	0	21		9	
warezmaster	20	20	4	0	20		1,602	
imap	12	12	1	0	12		1	
rootkit	10	10	0	2	10		13	
loadmodule	9	9	0	0	9		2	
ftp_write	8	8	0	1	8		3	
multihop	7	7	0	2	7		18	
phf	4	4	1	0	4		2	
perl	3	3	0	0	3		2	
spy	2	2	0	0	2		0	
snmpgetattack							7,741	
named							17	
xlock							9	
xsnoop							4	
sendmail							17	
saint							736	
apache2							794	
udpstorm							2	
mscan							1,053	
processtable							759	
ps							16	
worm							2	
httptunnel							158	
sqlattack							2	
snmpguess							2,406	
mailbomb							5,000	
xterm							13	
Total	**494,021**	**43,752**	**44,000**	**44,000**	**169,000**	**44,000**	**311,029**	**120,581**

There are 169 000 records in Set4 that contains all the attack types. This set has an adequate representation of all the service types, flags and protocol types for normal, smurf and Neptune types.

Once the training is complete, we store the number of features used, the number of neurons used, the weights of the neurons, some state information for each neuron such as label, confidence factor etc., in the state file. At test time we construct the test vector for each KDD Cup 1999 data record in the test set using the feature set that was used for the training. We used the test set labeled "corrected" from the KDD Cup 1999 benchmark set for the testing. The test set contains 17 additional attack types that are not present in any of the training set used. One motivation of using this set is that it has labels, and hence we can verify the accuracy of our detection scheme. Another incentive in using this test set is that the test set and the training set have no common elements.

C. Testing of the Single-Layer K-Map and Feature Vector Selection

As we have discussed, choosing the entire set of 41 features for the K-Map clustering may not be productive. First of all, it is computationally costly to do the normalization and vector multiplication for such a large vector. Even if we disregard the computational complexity, choosing the entire feature space will not yield a reasonable grouping of the records. In selecting a subset of the feature space, it is beneficial to take into account some domain knowledge from the field of intrusion detection. For instance, the ICMP protocol is some times used by malicious hackers to create many echo requests from a spoofed IP address that are then broadcast over a target network. If all the hosts on the victim network start sending echo reply to the sender, it can degrade the performance of the network. The smurf attack is an example of such a denial of service attack [5]. However we need to understand that a perfectly normal connection can also use ICMP protocol and echo request service to do a harmless ping operation. Our experiment shows that within the context of the KDD Cup 1999 data, the ICMP protocol and the echo request (ecr_i) service together with the number of bytes transmitted from source to destination can uniquely group all smurf attack records into a cluster.

Many scanning operations to glean information about active hosts and possible vulnerabilities in the network are also done using ICMP protocol and echo request service. For instance, one mode of $ipsweep$ scan uses ICMP protocol and eco_i service. In this case, the number of bytes transmitted from source to destination is usually 8 bytes. Another vulnerability scanner that has several manifestations is the $nmap$. When a UDP-based packet is sent to a device, the router sometimes fails to deliver the packet to the destination, either because the destination is temporarily unavailable or the destination address is nonexistent. In such cases a Network unreachable or Port unreachable ICMP message is returned. This is one mode of probing used by $nmap$. Another mode is through TCP protocol with half open SYN $(SH\ flag)$. Besides scanning activities that may be precursors to an attack, there are many other types of attacks that cause denial of service attacks such as teardrop, ping of death, back etc. The teardrop attack exploits weakness in the reassembly of packet fragments by creating fragments with overlapping offset fields. Attempts to reassemble such wrong fragments cause failures such as crashing, hanging or rebooting at the destination host. Other types of attacks need observation over a period of time to point to an anomalous activity. Then there are indications such as number of failed login attempts that point to probable intrusions. Thus, the different nature of features such as service, protocol, number of bytes from source to destination, the flag, number of wrong fragments are some of the important features that could be used in formulating the feature subspace for the K-Map.

For our experiments we identified different subsets of features that are good in detecting certain types of attacks. But we will soon see that a single such subset will not yield good detection rate at an acceptable false positive rate for a variety of attacks. A more comprehensive set of feature subsets that yielded good performance is listed in Table IV in Section III. These subsets were identified in account of domain knowledge and the nature of signal types (IP-protocol based, statistical, dynamical, etc.). Our approach of feature selection groups features in terms of their coherent relations (such as static, IP dumping vs. dynamic/time-related, temporal data traffic attributes) and mutual inclusiveness. The result of this approach is that the resulting clusters of a hierarchical K-Map could be more cohesive compared to that of random selection of feature groups. Three of the subsets that we used for single-layer K-map testing cases are listed as follows.

Feature set 1:

- Protocol, the protocol used such as ICMP, TCP, UDP
- Service, such as http, ftp, smtp, eco_i, ecr_i etc.
- SrcBytes, the number of bytes transmitted from source to destination

TABLE II
RESULTS FOR TESTS ON TEST SET 1 FOR SINGLE LAYER K-MAP WITH 36 NEURONS

Feature subset	Training data	%Detected	%False Positive
Feature set 1	Set 1	95.17	65.78
	Set 2	72.54	33.25
	Set 3	73.23	41.76
Feature set 2	Set 1	100	98.83
	Set 2	98.86	73.28
	Set 3	94.97	59.33
Feature set 3	Set 1	99.91	77.53
	Set 2	99.91	77.53
	Set 3	99.78	78.06

- DstBytes, the number of bytes transmitted from destination to source

Feature set 2:

- Protocol, the protocol used such as ICMP, TCP, UDP
- Service, such as http, ftp, smtp, eco_i, ecr_i etc.
- Flag
- Duration, the length of the connection
- srcBytes, the number of bytes transmitted from source to destination
- dstBytes, the number of bytes transmitted from destination to source
- wrongFragments, the number of wrong fragments encountered in reassembly.
- urgentPackets, the number of urgent packets
- count, the number of connections made to the same host in a given time interval
- sameHstSynErrRate, fraction of connections from the same host that had SYN errors in a specified time interval
- sameSvcSynErrRate, fraction of connections with the same host and same service that had syn errors.
- sameHstRejErrRate, fraction of connections from the same host that had REJ error.
- sameSvcRejErrRate, fraction of connections with the same host and same service that had REJ error.
- sameHstSameSvcRate, fraction of connections from the same host that used the same service.
- sameHstDiffSvcRate, fraction of connections from the same host that used different services.
- sameSvcDiffHstRate

Feature set 3:

- sameHstSynErrRate
- sameSvcSynErrRate
- sameHstRejErrRate
- sameSvcRejErrRate
- sameHstSameSvcRate
- sameHstDiffSvcRate
- sameSvcDiffHstRate

After selecting the training sets and feature subsets described above, we selected 36 neurons and a threshold of 0.000 012 5 for the first experiment. That is the training iterations are continued until number of changed mappings is less than 2. The results obtained for the test cases with respect to different feature sets and training data sets are given in Table II. Next we used 48 neurons instead of 36 and repeated the same tests. The results are shown in Table III.

It can be seen from Tables II and III that even though the detection rate is high, the false positive rate is unacceptably high too in most of the above test cases. Increasing the number of neurons has not reduced the false positives much either. In fact

TABLE III
RESULTS FOR TESTS ON TEST SET 1 FOR SINGLE LAYER K-MAP WITH 48 NEURONS

Feature Subset	Training Data	% Detected	%False Positive
Feature set 1	Set 1	98.21	82.32
	Set 2	72.53	32.18
	Set 3	73.23	40.14
Feature set 2	Set 1	100	98.69
	Set 2	86.71	62.21
	Set 3	94.05	62.35
Feature set 3	Set 1	99.9	76.39
	Set 2	99.52	75.47
	Set 3	99.7	76.85

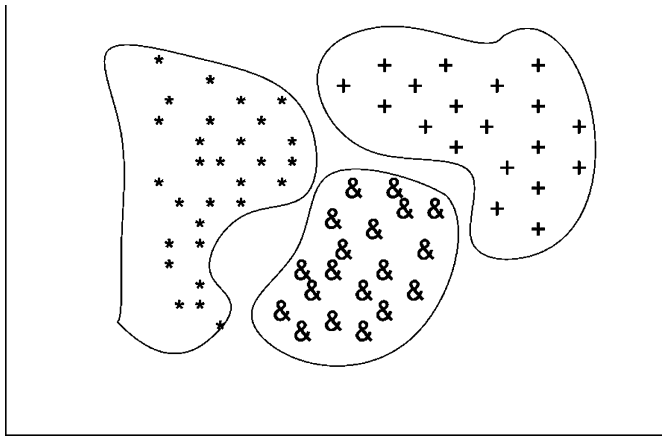

Fig. 3. Clusters formed by intersection of hyper cylinders.

the false positive rate increased in some cases when number of neurons was increased to 48. It indicates that a single-layer K-Map does not seem to be the ideal candidate for detecting a variety of attacks and at the same time having low false positive rate. On the other hand, it is also seen that some feature set has a better performance on a particular test case (with respect to a selective training set). More detailed observation into the test results reveals that some feature sets are more sensitive in detecting certain types of anomalies and some for other types. So next in our experimentations, we explored the feasibility of using a multilayer hierarchical winner-takes-all K-Map that takes advantage of the feature vector subdivisions for IDS.

III. HIERARCHICAL WINNER-TAKES-ALL K-MAP FOR ANOMALY DETECTION

In Section II, we explored the use of a single-layer K-Map as a classifier for KDD99 benchmark records. A conceptual interpretation is that the clusters in the case of single-layer K-Maps were modeled as hyper-spheres. In the case of a multilayer hierarchical winner-takes-all K-Map where mutually exclusive subsets of the feature vectors are fed to the multiple neuron layers, the resulting clusters are modeled in fact as intersections of hyper-cylinders. This enables the hierarchical K-Map to have a high-order nonlinear classifier modeling than the single-layer K-Map, see Fig. 3 for a graphical illustration of resulting clusters in two dimensions.

In the following, first we give a brief description of the hierarchical K-Map model that we implemented. Our model of hierarchical K-Map has three levels. Each level fits the description of a single-layer winner-takes-all K-Map. The number of neurons used in each level is tunable by the user. The subset of features

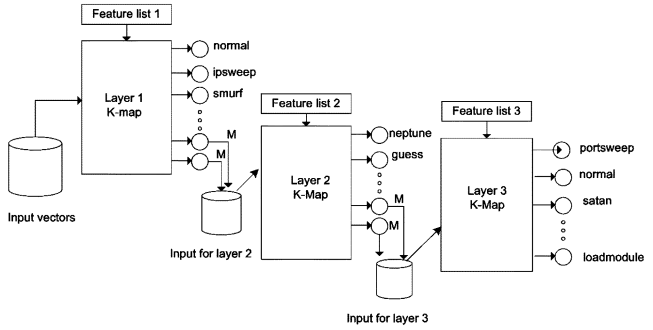

Fig. 4. Three-level hierarchical winner-takes-all map.

to be used for each level is also selectable by the user. Additionally, the threshold that determines the number of training iterations for each level is tunable using the graphical user interface. The organization of a three-level hierarchical K-Map is shown in Fig. 4, where M represents a cluster that has a heterogeneous mix of attack records and/or normal records.

A. Training and Testing Algorithms of Hierarchical K-Map

The training algorithm of the hierarchical K-Map has a graphical user interface that allows users to select the number of levels, and the number of neurons to be applied in each of the K-Map neuron levels. Users also use the graphical interface to select the feature subset, and set the training threshold for each level. The algorithm is presented as follows.

Algorithm Train_Hierarchical_K_Map

Step 1: Get the number of levels n from the user

Step 2: Read each KDD-99 record from the input file. Let Φ denote the set of input records. Extract the feature subsets **for** levels $1, 2, \ldots, n$.

Step 3: **for** each level $s \in \{1, 2, \ldots, n\}$ perform steps 4 to 5.

Step 4: if $\Phi \neq empty$ train level s map using the technique described in $algorithm\ train\_simple\_winner\_take\_all$.

Step 5: **for** each cluster i in level s

if cluster i is homogeneous

Label cluster i with the unique label of the records in this cluster

Set confidence factor **for** i, $C_i \leftarrow 1.0$

Let H denote the set of input records that mapped to homogeneous cluster i

$\Phi \leftarrow \Phi - H$

else begin

Compute A, the set of labels that mapped to i.

For each label X in cluster i, compute

$$F_i(X) = \left(\frac{X_i}{X_{\text{total}}}\right)\left(\frac{X_i}{N_i}\right)$$

Set the label **for** cluster i, $L(i) \leftarrow \alpha$ where

$$F_i(\alpha) = \max_{x \in A} F_i(x).$$

Set confidence factor

$$C_i \leftarrow \frac{\max F_i(x)}{\sum_{x \in A} F_i(x)}$$

end (else)

Step 6: Store the multilayer network parameters (# of neurons, weights, thresholds, mappings) in a state file.

We used the following algorithm to test the hierarchical K-Map. Here n denotes the number of levels of the trained SOM.

Algorithm Test_Hierarchical_K_Map

Step 1: Initialize the hierarchical K-Map with the parameters stored in a state file.

Step 2: **for** each KDD 99 Cup test record **do**

Construct the test vectors **for** levels 1 to n using the corresponding feature subspace.

$reliablyClassified \leftarrow false$

$level \leftarrow 1$

while $(level < n)$ and not $reliablyClassified$ **do**

Feed the test vector **for** level i to the ith level K-Map.

Look up the encapsulation of label, neuron number, confidence, etc. **for** the winner neuron.

$reliablyClassified \leftarrow (confidence\ equals\ 1)$ $or (label = "Undefined")$

$level \leftarrow level + 1$

end while

Step 3: Choose the label and corresponding confidence from the level that has highest confidence.

Step 4: Compute total number of false positives and detected anomalies

B. Computational Complexity

Computational complexity of the hierarchical K-Map algorithm can be derived in general as the following. Training is generally done offline and less frequently compared to detection. It is the computational cost at the detection phase that is more crucial to an intrusion detection system. Consider an L-level map with K neurons in each level. Let M denote the number

of features in a data record, and assume that each level uses the same number of features but no two levels have overlapping features, then the number of features at level i is M/L. As discussed in Section I, a single feature by itself is not a good indicator of anomaly. Hence, each level should have at least two features. Thus, L should be less than $(M/2)$. We thus have the weight matrix size of (K^*M/L) for each layer. For an input, finding the winner neuron is the most costly operation. The basic operation involved in this step is floating point multiplication. In choosing the winner neuron (as in the test phase) for an input at level i we need to perform $\left(K*(M/L)^2\right)$ multiplications. If the winner has a looked up confidence of 1.0 the input is not fed to any subsequent layers. In the worst case, an input will be fed to all L layers. In that case, there are $L\left(K*(M/L)^2\right) = K^*M^2/L$ basic multiplication operations performed. Let n be the number of input records, then the computational complexity for the hierarchical K-Map in the test phase is $O(nKM^2)$, which is the same as for a single-layer K-Map. In the traditional sense of computational complexity, as the number of input record $n \to \infty$, M^2/L is a small constant. Hence, the computational complexity in the detection phase of the hierarchical K-Map is $O(nK)$.

For the training phase, the computational complexity is increased due to the number of training iterations performed. In general, the number of training iterations is proportional to the number of training records n, the number of neurons K, and the number of features M. The number of training iterations is determined by the distributive nature of clusters in the data records but it will never exceed n. In the worst case there are (KM^2n^2/L) basic operations performed. If we take into account the fact that M^2/L is less than M^2 the number of iterations is less than that in the single-layer K-Map cases that uses all M features. This is where the computational efficiency of the hierarchical K-Map comes from in addition to the high precision of clustering, though it is very difficult to give a quantitative measurement of the computational complexity due to the fact that it depends on the quality of the cluster distributions of the training data set. We present our experiment results in Section III-C.

C. Tests and Results

In addition to the four training sets listed in Section II, we also created a Set5 with 44 000 randomly selected records from the 10% training set in testing the hierarchical K-Map. In Set5, we restrict the record types to neptune, portsweep, satan and normal (Table I). For testing, we use Test set 1 as well as Test set 2 (Table I) a subset of test set 1 that contains only records of type normal, neptune, satan, and portsweep. We conducted experiments with several different combinations of feature sets for the different layers of the hierarchical K-Map. Five combinations of these feature subsets used in our experiments are listed in Table IV. The motivation behind choosing the first four feature subsets as in Table IV is to get a maximal first level separation of normal and anomalous records. Subsequent levels will further refine the classification to more specific types of anomalies. The features listed under level 1 for each of the first four combinations help to filter out maximum normal records from anomalous records, thus reducing computational cost. No such maximal filtering is expected from the fifth combination.

TABLE IV
SOME FEATURE SUBSET COMBINATIONS FOR MULTILAYER HIERARCHICAL K-MAP TESTS

Combination	First Level Features	Second Level Features	Third Level Features
1	dstBytes	Fragments	sameSvcsynErrRate
	srcBytes	Flag	sameHstSameSvcRate
	Protocol	numRootAccess	sameHstDiffSvcRate
	Service	Logged status	sameSvcDiffHstRate
		Hot	dstHstSvcDiffHstRate
		Failed logins	dstHstSameSvcRate
		numFileCreation	dstHstSameSrcPortRate
			Count
			sameHstSynErrRate
2	dstBytes	Service	sameSvcSynErrRate
	srcBytes	Flag	sameHstSameSvcRate
	Protocol	numRootAccess	sameHstDiffSvcRate
	Fragments	dstHstSameSvcRate	sameSvcDiffHstRate
		dstHstSameSrcPortRate	dstHstSvcDiffHstRate
		Count	Logged status
		sameHstSynErrRate	Hot
			Failed logins
			numFileCreation
3	dstBytes	Flag	Hot
	srcBytes	numRootAccess	Failed logins
	Protocol	Logged status	numFileCreation
	Service	dstHstSameSvcRate	sameSvcSynErrRate
	Fragments	dstHstSameSrcPortRate	sameHstSameSvcRate
		Count	sameHstDiffSvcRate
		sameHstSynErrRate	sameSvcDiffHstRate
			dstHstSvcDiffHstRate
			sameHstRejErrRate
4	dstBytes	guestLogin	Failed logins
	srcBytes	Flag	numFileCreation
	Protocol	Hot	numRootAccess
	Service	Logged status	sameSvcSynErrRate
	Count	dstHstSvcDiffHstRate	sameHstSameSvcRate
		dstHstSameSvcRate	sameHstDiffSvcRate
		dstHstSameSrcPortRate	sameSvcDiffHstRate
		Count	sameHstRejErrRate
		sameHstSynErrRate	Fragments
5	sameHstSynErrRate	duration	hot
	sameSvcSynErrRate	protocol	Failed Logins
	sameHstRejErrRate	service	numCompromised
	sameSvcRejErrRate	flag	numRootAccess
	sameHstSameSvcRate	srcBytes	numFileCreation
	sameHstDiffSvcRate	dstBytes	numShells
	SameSvcDiffHstRate	landStatus	numAccessFiles
		Fragments	numOutBoundCommands
		Urgent packets	

We trained the hierarchical K-Map using each of the four training sets described in Section II-B for several combinations of features, number of levels and number of neurons. Next we tested the hierarchical K-Map with the test set 1 for each of the training cases. Note that test set 1 contains 17 additional attack types that are not present in any of the training sets.

A summary of sixty such test cases is presented in Table V. From Table V, it can be seen that detection and false positives are consistently better for the detector when trained with set 4 for each of the first four feature combinations. The presence of all possible protocols and services for normal records in training data seems to be the reason for the better performance in each of these cases. As for the effect of increasing the number of neurons per layer, the detection rate tends to increase for 48 neurons and then it decreases for 72 neurons. Further increasing the number of neurons reduced the detection rate. The percentage of false positives tends to decrease by increasing the number of neurons. However, the improvement in false positives compared to the decrease in detection rate is not significant beyond 72 neurons. The percentages of detection for each attack type for 15 test cases are summarized in Tables VII–IX. Table VII shows the results when 36 neurons were used in each layer. Tables VIII and IX represent the cases for 48 neurons per layer and 72 neurons per layer respectively. The new attacks types are shown in bold. The number of new attacks detectable and the percentage

TABLE V
SUMMARY OF RESULTS FOR 60 TEST CASES FOR A THREE-LEVEL HIERARCHICAL K-MAP

Feature Combination	Training Set	% Anomaly Detection			% False Positive		
		36 neurons	48 neurons	72 neurons	36 neurons	48 neurons	72 neurons
Combination 1	Set 1	95.37	95.36	94.8	10.36	11.94	11.00
	Set 2	70.07	72.08	70.12	9.64	10.22	9.19
	Set 3	72.08	72.53	70.82	10.24	10.40	10.02
	Set 4	70.3	70.75	70.68	2.87	3.35	2.75
Combination 2	Set 1	90.73	87.74	3.99	4.87	4.92	9.61
	Set 2	94.7	94.39	24.72	27.09	24.74	1.22
	Set 3	95.11	95.02	24.93	27.38	24.38	3.24
	Set 4	92.02	92.56	90.94	3.63	3.00	3.43
Combination 3	Set 1	94.88	95.42	94.79	12.93	14.35	11.06
	Set 2	94.94	94.51	90.44	25.68	24.25	10.18
	Set 3	93.52	93.32	92.59	10.46	12.98	10.82
	Set 4	93.35	93.46	93.27	3.56	3.99	3.41
Combination 4	Set 1	47.69	47.78	77.86	2.49	2.40	1.52
	Set 2	93.26	93.05	60.69	19.62	17.16	3.04
	Set 3	91.97	91.55	77.45	6.84	7.66	2.00
	Set 4	78.18	91.37	91.29	2.19	2.92	2.76
Combination 5	Set 1	94.9	95.07	95.13	12.09	13.77	12.30
	Set 2	93.36	96.29	96.42	14.63	29.69	29.51
	Set 3	94.41	94.31	97.70	14.74	13.22	24.97
	Set 4	79.07	94.92	94.67	13.23	13.16	11.36

TABLE VI
RESULTS OBTAINED FOR TEST SET 2

Feature combination	%Detected	%False positive
1	99.63	0.34
2	99.17	0.45
3	99.6	0.35
4	99.51	1.41

of detection by type for individual attacks are comparable for the remaining 45 test cases. Lastly, we trained the three-level hierarchical K-MAP with trainingSet4 and tested on test set 2. For this training and testing, we used 48 neurons in each level. The results for each of the four combinations of features are listed in Table VI. The results show that the false positive rates are significantly reduced in the hierarchical K-Map than that in the single-layer K-Map on all test cases.

On examining the results in Tables VII–IX, it is clear that not all attack types can be effectively detected by a specific feature combination. Some feature combinations are excellent in capturing certain types of attacks. We explored the reasons for the lack luster detection rates for buffer_overflow, guess_passwd, and xsnoop. We noticed that for the subset of features considered, the feature vectors of those anomalies either closely matched that of normal records or they were closely intertwined in a cluster of normal records. In the case of xsnoop, the feature vectors closely matched that of a normal record. See Fig. 5 for a graphical illustration of these two situations. Here, the circle represents a data sample corresponding to the feature vector of a normal record and the star represents an anomaly.

IV. RELATED WORK AND CONCLUSIONS

In 1990, Fox *et al.* [6] proposed the use of K-Map to detect the presence of virus in a multiuser machine. They applied the K-Map to learn the characteristics of normal system activity. Variations from the normal profile were considered as indications of the presence of a virus. Cannady [7] used K-Map in network misuse detection. The approach used a hybrid neural network in which the output of a K-Map provided input to a hybrid feed-forward neural network. Rhodes *et al.* [8] analyzed the potential of K-Map to narrow the envelope of intrusion behavior that would not be caught by a detection system. Jirapummin *et al.* [9] used a hybrid neural network model that employed the

TABLE VII
DETECTION BY ATTACK TYPE FOR 36 NEURONS PER LAYER (NEW ATTACK TYPES IN BOLD)

Attack type	Combination 1	Combination 2	Combination 3	Combination 4	Combination 5
normal	97.13%	96.37%	96.44%	97.81%	86.77%
snmpgetattack.	**0.01%**	**0.49%**	**0.00%**	**0.00%**	**0.10%**
named.	**70.59%**	**70.59%**	**64.71%**	**29.41%**	**52.94%**
xlock.	**44.44%**	**44.44%**	**44.44%**	**0.00%**	**66.67%**
smurf.	100.00%	99.99%	100.00%	80.10%	100.00%
ipsweep.	99.67%	100.00%	99.67%	100.00%	99.67%
multihop.	44.44%	38.89%	33.33%	5.56%	27.78%
xsnoop.	**0.00%**	**25.00%**	**25.00%**	**25.00%**	**0.00%**
sendmail.	**23.53%**	**11.76%**	**35.29%**	**0.00%**	**17.65%**
guess_passwd.	0.02%	0.11%	0.21%	0.69%	13.97%
saint.	**89.27%**	**98.37%**	**97.69%**	**98.37%**	**88.72%**
buffer_overflow.	13.64%	22.73%	36.36%	4.55%	22.73%
portsweep.	13.56%	98.87%	95.76%	95.48%	66.95%
pod.	95.40%	100.00%	98.85%	87.36%	98.85%
apache2.	**95.47%**	**28.34%**	**83.50%**	**89.29%**	**98.36%**
phf.	50.00%	100.00%	100.00%	0.00%	50.00%
udpstorm.	**0.00%**	**100.00%**	**0.00%**	**0.00%**	**100.00%**
warezmaster.	65.42%	40.26%	62.05%	55.31%	65.36%
perl.	0.00%	0.00%	50.00%	0.00%	50.00%
satan.	92.10%	99.94%	99.20%	99.76%	91.79%
xterm.	**38.46%**	**53.85%**	**30.77%**	**0.00%**	**30.77%**
mscan.	**49.76%**	**40.55%**	**66.00%**	**47.96%**	**56.51%**
processtable.	**40.05%**	**17.92%**	**11.20%**	**0.00%**	**2.11%**
ps.	**6.25%**	**6.25%**	**6.25%**	**18.75%**	**6.25%**
nmap.	98.81%	100.00%	4.76%	4.76%	5.95%
rootkit.	61.54%	61.54%	46.15%	0.00%	46.15%
neptune.	5.61%	99.96%	99.78%	99.98%	33.74%
loadmodule.	0.00%	0.00%	0.00%	0.00%	0.00%
imap.	100.00%	100.00%	100.00%	100.00%	100.00%
back.	99.18%	98.27%	99.73%	89.34%	90.80%
httptunnel.	**91.77%**	**89.87%**	**91.77%**	**85.44%**	**93.67%**
worm.	**0.00%**	**0.00%**	**100.00%**	**0.00%**	**0.00%**
mailbomb.	**0.00%**	**3.34%**	**99.70%**	**0.00%**	**99.78%**
ftp_write.	33.33%	33.33%	33.33%	0.00%	66.67%
teardrop.	75.00%	100.00%	100.00%	66.67%	100.00%
land.	100.00%	100.00%	100.00%	100.00%	100.00%
sqlattack.	**0.00%**	**0.00%**	**0.00%**	**100.00%**	**100.00%**
snmpguess.	**87.28%**	**94.72%**	**0.29%**	**0.33%**	**95.93%**

TABLE VIII
DETECTION BY ATTACK TYPE FOR 48 NEURONS PER LAYER (NEW ATTACKS IN BOLD)

Attack type	Combination 1	Combination 2	Combination 3	Combination 4	Combination 5
normal	96.65%	97.00%	96.01%	97.08%	86.84%
snmpgetattack.	**0.01%**	**0.26%**	**0.00%**	**0.00%**	**0.10%**
named.	**70.59%**	**52.94%**	**70.59%**	**41.18%**	**58.82%**
xlock.	**55.56%**	**55.56%**	**44.44%**	**0.00%**	**66.67%**
smurf.	99.99%	99.99%	99.99%	100.00%	100.00%
ipsweep.	99.02%	100.00%	98.69%	99.67%	99.67%
multihop.	38.89%	55.56%	27.78%	27.78%	27.78%
xsnoop.	**0.00%**	**25.00%**	**0.00%**	**50.00%**	**0.00%**
sendmail.	**23.53%**	**11.76%**	**5.88%**	**35.29%**	**17.65%**
guess_passwd.	15.20%	16.26%	0.73%	0.69%	15.94%
saint.	**88.86%**	**98.37%**	**97.42%**	**98.51%**	**98.10%**
buffer_overflow.	13.64%	18.18%	18.18%	9.09%	22.73%
portsweep.	14.97%	96.33%	93.79%	83.90%	46.61%
pod.	95.40%	100.00%	98.85%	94.25%	98.85%
apache2.	**80.98%**	**30.60%**	**84.63%**	**84.51%**	**98.24%**
phf.	50.00%	100.00%	100.00%	0.00%	50.00%
udpstorm.	**0.00%**	**100.00%**	**0.00%**	**0.00%**	**0.00%**
warezmaster.	99.50%	93.51%	66.29%	63.80%	98.31%
perl.	0.00%	0.00%	0.00%	0.00%	50.00%
satan.	92.10%	100.00%	99.20%	99.76%	99.76%
xterm.	**46.15%**	**46.15%**	**15.38%**	**15.38%**	**30.77%**
mscan.	**22.22%**	**55.84%**	**67.24%**	**66.57%**	**81.10%**
processtable.	**55.86%**	**99.60%**	**38.08%**	**0.79%**	**34.52%**
ps.	**6.25%**	**0.00%**	**6.25%**	**37.50%**	**6.25%**
nmap.	98.81%	100.00%	5.95%	4.76%	100.00%
rootkit.	61.54%	30.77%	53.85%	15.38%	46.15%
neptune.	5.62%	99.98%	99.76%	99.98%	99.99%
loadmodule.	0.00%	0.00%	0.00%	0.00%	0.00%
imap.	100.00%	0.00%	100.00%	100.00%	100.00%
back.	99.18%	20.22%	99.64%	99.18%	86.98%
httptunnel.	**89.24%**	**92.41%**	**91.14%**	**87.34%**	**93.04%**
worm.	**0.00%**	**0.00%**	**100.00%**	**0.00%**	**0.00%**
mailbomb.	**0.00%**	**0.26%**	**99.90%**	**0.00%**	**99.78%**
ftp_write.	33.33%	0.00%	0.00%	33.33%	33.33%
teardrop.	100.00%	100.00%	83.33%	66.67%	100.00%
land.	100.00%	100.00%	100.00%	100.00%	100.00%
sqlattack.	**0.00%**	**0.00%**	**0.00%**	**100.00%**	**100.00%**
snmpguess.	**95.26%**	**95.34%**	**0.17%**	**0.29%**	**95.93%**

TABLE IX
DETECTION BY ATTACK TYPE FOR 72 NEURONS PER LAYER
(NEW ATTACKS IN BOLD)

Attack type	Combination 1	Combination 2	Combination 3	Combination 4	Combination 5
normal	97.25%	96.57%	96.59%	97.24%	88.64%
snmpgetattack.	**0.01%**	**0.26%**	**0.00%**	**0.00%**	**0.10%**
named.	**70.59%**	**58.82%**	**70.59%**	**47.06%**	**58.82%**
xlock.	**55.56%**	**0.00%**	**44.44%**	**22.22%**	**55.56%**
smurf.	99.98%	98.27%	100.00%	99.99%	100.00%
ipsweep.	100.00%	99.35%	99.67%	99.35%	100.00%
multihop.	16.67%	50.00%	27.78%	16.67%	38.89%
xsnoop.	**0.00%**	**0.00%**	**0.00%**	**50.00%**	**0.00%**
sendmail.	**23.53%**	**11.76%**	**5.88%**	**35.29%**	**17.65%**
guess_passwd.	15.27%	16.35%	0.66%	0.80%	16.26%
saint.	**74.18%**	**96.88%**	**83.42%**	**97.01%**	**89.95%**
buffer_overflow.	0.00%	22.73%	22.73%	22.73%	22.73%
portsweep.	23.45%	98.02%	93.79%	84.46%	14.69%
pod.	96.55%	78.16%	98.85%	93.10%	98.85%
apache2.	**76.32%**	**23.55%**	**74.18%**	**85.52%**	**96.73%**
phf.	50.00%	50.00%	100.00%	50.00%	0.00%
udpstorm.	**0.00%**	**100.00%**	**0.00%**	**0.00%**	**0.00%**
warezmaster.	97.94%	39.08%	66.48%	57.55%	98.38%
perl.	0.00%	0.00%	0.00%	0.00%	50.00%
satan.	80.71%	99.69%	88.43%	99.08%	94.12%
xterm.	**30.77%**	**38.46%**	**30.77%**	**23.08%**	**23.08%**
mscan.	**42.64%**	**42.83%**	**64.29%**	**61.44%**	**63.82%**
processtable.	**51.65%**	**37.02%**	**37.29%**	**0.53%**	**33.99%**
ps.	**6.25%**	**6.25%**	**6.25%**	**37.50%**	**6.25%**
nmap.	100.00%	100.00%	5.95%	4.76%	98.81%
rootkit.	61.54%	61.54%	61.54%	15.38%	46.15%
neptune.	5.62%	99.79%	99.97%	99.99%	99.88%
loadmodule.	0.00%	0.00%	0.00%	0.00%	0.00%
imap.	100.00%	0.00%	100.00%	100.00%	100.00%
back.	99.00%	64.75%	95.99%	94.17%	85.88%
httptunnel.	**91.77%**	**91.77%**	**91.77%**	**87.34%**	**20.89%**
worm.	**0.00%**	**0.00%**	**100.00%**	**0.00%**	**0.00%**
mailbomb.	**0.00%**	**0.00%**	**96.40%**	**0.00%**	**99.78%**
ftp_write.	33.33%	33.33%	0.00%	33.33%	66.67%
teardrop.	83.33%	100.00%	100.00%	91.67%	100.00%
land.	100.00%	100.00%	100.00%	100.00%	100.00%
sqlattack.	**0.00%**	**0.00%**	**0.00%**	**100.00%**	**100.00%**
snmpguess.	**95.26%**	**94.56%**	**0.17%**	**0.29%**	**95.93%**

output weight information from a K-Map fed to a resilient propagation neural network (*RPROP*) to detect TCP SYN flooding and port scan attacks. They used a Gaussian neighborhood function and a cluster matching function to realize the K-Map, and also used the KDD Cup 1999 data for training and testing. The hybrid model was made up of a 1,234 unit K-Map followed by a three-layer RPROP network of 70, 12, and fourneurons, respectively. Sigmoid functions were used for each level of the RPROP network. The focus was only on three different attacks, the Neptune (a SYN flood attack), the satan probe, and the portscan probe. They achieved 90% detection rate for satan attacks at 4.5% false positive rate, 97.9% detection rate for portsweep at 4.19% false positive rate, and 99.72% detection rate for neptune attacks at 0.06% false positive rate.

At Dalhousie University, Heywood *et al.* used self-organizing maps to perform host-based intrusion detection and network based intrusion detection [10]. In both cases, they used the K-Map Toolbox from MATLAB in realizing the self-organizing maps. The network based IDS prototype used the pre-processed KDD Cup 1999 data for training and testing. They used a hierarchical neural network approach based on K-Map and potential function clustering [10]. Six basic features from the KDD Cup 1999 records namely duration, protocol, service, flag, destination and source were used. At level 1, separate K-Maps were used for each of these six features. The second level K-Map combined the features detected by the first level into a single view. Potential function clustering was used to quantize the number of inputs seen by the second layer. A Gaussian hexagonal neighborhood is used. They achieved a 89% detection rate at a false positive rate of 4.6%.

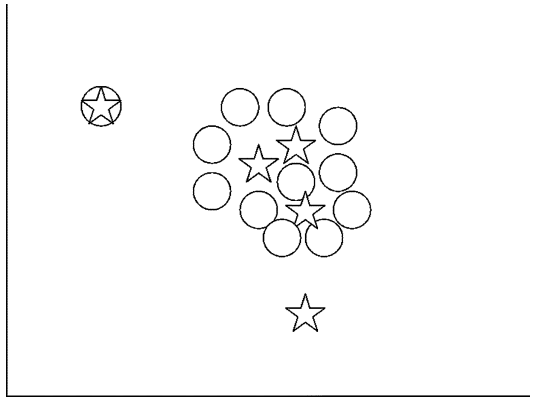

Fig. 5. Cases where anomalies intertwine with normal clusters.

The novelty of our work lies in the following aspects. First, our approach uses the simplest form of Kohonen self-organizing map with no neighborhood functions or transfer functions. It is a very low-cost implementation in terms of computational complexity. When coming to the stage of on-line network traffic monitoring and real-time intrusion detection, it is only the detection phase that needs to be involved. In the detection phase, there is no iterative process involved as in the case of training phase. Thus, each record (network connection from initiation to termination) will pass in a single pulse (in terms of parallel processing characteristics of neural network) of computation, with a very minor delay of traffic. Secondly the implement of our hierarchical K-Map allows selection of different combinations of feature subsets with a wide range of selection for number of neurons used in each level as well as for the training threshold. Thirdly, the hierarchical structure we use is quite different from that in [9] and [10] in the way that each level uses a different subset (mutually exclusive) of features to construct the input vector for that level. Test vectors that are reliably classified (a confidence factor of 1.0) in level i are not tested further in subsequent levels. When reliable classification is not achieved at level i, test continues on subsequent levels until a reliable classification is achieved or until all three levels are covered. Based on the tests we conducted, we have achieved detection rates between 90.94% and 93.46% at false positive rates between 2.19% and 3.99% for feature combinations 2, 3, and 4. Three of these results were achieved using 72 neurons in each level, another three using 48 neurons in each level and two cases using 36 neurons in each level. These are better than those achieved in [10]. We were able to achieve 99.63% detection rate at a false positive rate of 0.34% when the training and testing were limited to the three attack types (Table VI). We believe that it is possible to improve these results further by conducting more tests with proper combination of feature subsets and K-Map settings.

ACKNOWLEDGMENT

One of the authors, S. Sarasamma, thanks R. Gilbert, Northrop Grumman Mission Systems for pointing her to the project that leads to this work.

REFERENCES

[1] W. Lee, S. Stolfo, and K. Mok, "A data mining framework for building intrusion detection models," in *Proc. 1999 IEEE Symp. Security and Privacy*, 1999, pp. 120–132.

[2] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan, "Cost-based modeling and evaluation for fraud and intrusion detection: Results from the JAM project," in *Proc. DARPA Information Survivabilty Conf. and Expo.*, vol. II, 2000, pp. 130–144.

[3] S. Stolfo *et al.*. (2002) The Third International Knowledge Discovery and Data Mining Tools Competition. [Online]. Available: http://kdd.ics.uci.edu/databases/kddCup99/kddCup99.html

[4] P. Wasserman, *Neural Computing Theory and Practice*. New York: Van Nostrand, 1989.

[5] S. Northcut and J. Novak, *Network Intrusion Detection*, 3rd ed. Indianapolis, IN: New Riders, 2002.

[6] K. Fox, R. Henning, and J. Reed, "A neural network approach toward intrusion detection," in *Proc. 13th Nat. Computer Security Conf.*, Washington, DC, 1990.

[7] J. Cannady and J. Mahaffey, "The application of artificial intelligence to misuse detection," in *Proc. 1st Recent Advances in Intrusion Detection (RAID) Conf.*, Louvain-la-Neuve, Belgium, 1998.

[8] B. Rhodes, J. Mahaffey, and J. Cannady, "Multiple self-organizing maps for intrusion detection," in *Proc. 23rd Nat. Information Systems Security Conf.*, Baltimore, MD, Oct. 2000.

[9] C. Jirapummin, N. Wattanapongsakorn, and P. Kanthamanon. Hybrid Neural Networks for Intrusion Detection Systems. [Online]. Available: http://dbvis.fmi.uni-konstanz-de/members/panse/seminar_ws0203/

[10] H. Kayacik, A. Zincir-Heywood, and M. Heywood, "On the capability of an SOM based intrusion detection system," in *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN'03)*, pp. 1808–1813.

[11] W. Lee, S. Stolfo, P. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang, "Real time data mining-based intrusion detection," in *Proc. 2nd DARPA Information Survivability Conf. and Expo.*, vol. 1, 2001, pp. 89–100.

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

[13] R. P. LippMann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation," in *Proc. DARPA Information Survivability Conf. and Expo.*, vol. 2, 2000, pp. 12–26.

[14] 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. Inform. Syst. Security*, vol. 3, pp. 262–294, 2000.

[15] R. A. Maxion and K. M. C. Tan, "Benchmarking Anomaly-based detection systems," in *Proc. 1st Int. Conf. Dependable Systems and Networks*, New York, pp. 623–630.

[16] ——, "Anomaly detection in embedded systems," *IEEE Trans. Comput.*, vol. 51, no. 2, pp. 108–120, Feb. 2002.

[17] K. M. C. Tan and R. A. Maxion, "Determining the operational limits of an anomaly-based intrusion detector," *IEEE J. Select. Areas Commun.*, vol. 21, no. 1, pp. 96–110, Jan. 2003.

[18] T. Kohonen, *Self-Organizing Maps*, 3rd extended ed, ser. Information Sciences. Berlin, Germany: Springer, 2001, vol. 30.

[19] H. Ying, T. Feng, J. Cao, X. Ding, and Y. Zhou, "Research on some problems in the Kohonen SOM algorithm," in *Proc. Int. Conf. Machine Learning and Cybernetics*, vol. 3, Nov. 2002, pp. 1279–1282.

[20] R. A. Kemmerer and G. Vigna, "Intrusion detection: A brief history and overview," *Computer*, vol. 35, no. 4, pp. 27–30, Apr. 2002.

[21] S. Cho, "Incorporating soft computing techniques into a probabilistic intrusion detection system," *IEEE Trans. Syst., Man, Cybern.*, pt. C, vol. 32, no. 2, pp. 154–160, May 2002.

[22] N. Ye, S. M. Emran, Q. Chen, and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Trans. Comput.*, vol. 51, no. 7, pp. 810–820, Jul. 2002.

[23] M. Hagenbuchner, A. Sperduti, and A. Tsoi, "A self-organizing map for adaptive processing of structured data," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 491–505, May 2003.

[24] H. Yin, "ViSOM—A novel method for multivariate data projection and structure visualization," *IEEE Trans. Neural Netw.*, vol. 13, no. 1, pp. 237–243, Jan. 2002.

[25] C. Manikopoulos and S. Papavassiliou, "Network intrusion and fault detection: A statistical anomaly approach," *IEEE Commun. Mag.*, vol. 40, no. 10, pp. 76–82, Oct. 2002.

[26] N. Ye, Y. Zhang, and C. M. Borror, "Robustness of the Markov-chain model for cyber-attack detection," *IEEE Trans. Rel.*, vol. 53, no. 1, pp. 116–123, Mar. 2004.

[27] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *Proc. 2003 Symp. Applications and the Internet*, Jan. 2003, pp. 209–216.

Suseela T. Sarasamma received the M.Eng. degree in electrical and computer engineering from Concordia University, Montreal, QC, Canada, in 1991 and the Ph.D. degree in computer science from the University of Nebraska at Lincoln in 1996.

She is a Senior Software Engineer at Northrop Grumman Mission Systems, Bellevue, NE. Her current interests are in the design and development of algorithms of scientific nature for practical applications. Some specific areas are global weather prediction, network intrusion detection, and data fusion.

Dr. Sarasamma has been a member of the ACM since 1993.

Qiuming A. Zhu received the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1986.

He is a Professor of computer science at the University of Nebraska at Omaha. His postdoctoral research was carred out in the Center for Computer Aids for Industrial Productivity, Rutgers University, New Brunswick, NJ. He was an Assistant Professor of computer science and engineering at the Oakland University, Oakland, MI, from 1986 to 1990. His research interests include digital image processing and computer vision, pattern recognition, neural networks, multi-agent software systems, and artificial intelligence applications in science and engineering.

Julie Huff is currently pursuing the M. S. degree in pathology/bio-informatics at the University of Nebraska Medical Center, Omaha.

She is a Senior Systems Architect with Northrop Grumman Mission Systems, Bellevue, NE, where she has lead research and development teams in a number of areas, including data discovery, dynamic agent architectures, network data mining, and an embedded guard-on-a-card project. She is one of the originators of the Security Kinetix patent developed for event dissemination and response in tactical environments.