University of Nebraska Omaha

University of Nebraska at Omaha
DigitalCommons@UNO

Computer Science Faculty Publications | Department of Computer Science

8-2006

Min–Max Hyperellipsoidal Clustering for Anomaly Detection in Network Security

Suseela T. Sarasamma
University of Nebraska at Omaha

Qiuming Zhu
University of Nebraska at Omaha, qzhu@unomaha.edu

Follow this and additional works at: http://digitalcommons.unomaha.edu/compscifacpub

 Part of the Computer Sciences Commons

Recommended Citation

Sarasamma, Suseela T. and Zhu, Qiuming, "Min–Max Hyperellipsoidal Clustering for Anomaly Detection in Network Security" (2006). *Computer Science Faculty Publications*. Paper 31.
http://digitalcommons.unomaha.edu/compscifacpub/31

This Article is brought to you for free and open access by the Department of Computer Science at DigitalCommons@UNO. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

Dr. C.C. and Mabel L. CRISS LIBRARY

Min–Max Hyperellipsoidal Clustering for Anomaly Detection in Network Security

Suseela T. Sarasamma and Qiuming A. Zhu, *Senior Member, IEEE*

***Abstract*—A novel hyperellipsoidal clustering technique is presented for an intrusion-detection system in network security. Hyperellipsoidal clusters toward maximum intracluster similarity and minimum intercluster similarity are generated from training data sets. The novelty of the technique lies in the fact that the parameters needed to construct higher order data models in general multivariate Gaussian functions are incrementally derived from the data sets using accretive processes. The technique is implemented in a feedforward neural network that uses a Gaussian radial basis function as the model generator. An evaluation based on the inclusiveness and exclusiveness of samples with respect to specific criteria is applied to accretively learn the output clusters of the neural network. One significant advantage of this is its ability to detect individual anomaly types that are hard to detect with other anomaly-detection schemes. Applying this technique, several feature subsets of the tcptrace network-connection records that give above 95% detection at false-positive rates below 5% were identified.**

***Index Terms*—Accretive construction, anomaly detection, confidence measurement, hyperellipsoidal clustering, neural networks, self-organizing map (SOM).**

I. Introduction

With the advent of high-speed interconnection networks, it has become easier to share information between organizations, within organizations, and between individuals. What used to take tons of paper to store information is now stored on high-speed storage devices such as compact disk recordable/rewriteable (CD-R/RW), digital versatile disc (DVD), storage arrays, etc. The availability of such high-speed mass-storage devices and reliable interconnection networks for information sharing has transformed modern offices and businesses. Uninterrupted communication networks and computing facilities have become a necessity of modern life. With its increasing importance in day-to-day life, network-based computing systems have become a frequent target for attackers, hobbyists, and criminals alike. These attacks are often costly in terms of lost time and financial loss.

Early detection and possible preventive measures are crucial in thwarting costly network intrusions. Research on techniques to identify intrusions in network-based computer systems has contributed to some intrusion-detection systems (IDSs). These IDS fall into three categories: signature-based intrusion-detection systems, anomaly-based intrusion-detection systems, and specifications-based intrusion-detection systems.

Signature-based intrusion-detection systems are normally known as misuse-detection systems. Misuse-detection systems apply a rule-based approach that uses stored signatures of known intrusion instances to detect attacks. The Snort system is an example of a signature-based IDS [34]. These IDSs are highly successful in accurately detecting previously known attacks. However, they fail to detect variants of known attacks whose signatures are not stored. They also fail in detecting new attacks, whose signatures are not known. When new attacks are discovered, the signature database has to be manually updated. The network will be vulnerable to the newly discovered attacks until the updated signature database is in effect. Another approach used for the intrusion-detection systems is the anomaly-detection approach. Anomaly-detection techniques are also known as novelty-detection techniques. Some of the earlier research in anomaly detection evolved from statistical outlier-detection techniques. A survey of various outlier-detection techniques from statistical techniques to machine-learning approaches can be found in [31]. Marsland provides a comprehensive survey of the various novelty (anomaly)-detection techniques including those using self-organizing maps (SOMs) in [33].

In the anomaly-based detection approach, a profile of what is perceived as normal behavior is first established. Deviants from the normal profile are considered as anomalies or potential attacks. In some cases, normal operations that exhibit behavior adherent to unseen mode of operation are detected as anomalies. Such cases of false detection of normal operations as anomalous operations are termed false positives. Unlike signature-based detection systems, there are no exact templates to match an unknown event. The merit of an anomaly-detection scheme is the absence of an enormous signature database. The hallmark of a good anomaly-based approach is a very high detection rate at a very low false-positive rate. Anomaly-detection techniques can be further broadly classified into two categories: 1) supervised anomaly detection and 2) unsupervised anomaly detection.

Supervised anomaly-detection approaches usually need a set of purely normal data with which a model is trained [14]. New data is checked to see how well it fits into the trained model. Presence of anomalies in the training data can hamper the detection of such events in future. The need for purely normal training data is somewhat restrictive. Some techniques

Manuscript received March 30, 2005; revised August 27, 2005 and December 1, 2005. This work was supported in part by the Air Force Research Laboratory, Rome, New York and in part by the Advanced Research Development Agency under Grant F30602-03-C-0247. This paper was recommended by Associate Editor M. Obaidat.

S. T. Sarasamma is with the Northrop Grumman Mission Systems, Bellevue, NE 68023 USA (e-mail: suseela_ts@yahoo.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).

used in supervised anomaly-detection schemes are support-vector machine (SVM) classification, statistical methods to determine sequences that occur more frequently in intrusion data, decision trees, ensemble-based approaches, and machine-learning approaches. Lane and Brodley addressed the problem of anomaly detection as one of learning to characterize the behaviors of an individual, a system, or a network in terms of temporal sequences of discrete data [18]. Barbara *et al.* at George Mason University developed the ADAM IDS, a supervised anomaly-detection system. To enhance the system's capability of detecting new attacks and to reduce the false-alarm rate as much as possible, they applied a pseudo-Bayes estimator method according to the prior and posterior probabilities of new attacks [15]. Another well-known product in this category is the event monitoring enabling responses to anomalous live disturbances (EMERALD) system developed at SRI, which combines signature-based analysis using production-based expert system toolset (PBEST) rules for misuse detection and statistical-profile-based anomaly detection [27]–[29]. Supervised anomaly-detection tasks can use unsupervised algorithms, especially when only normal data are available, as demonstrated in [35].

Unsupervised anomaly-detection schemes are less restrictive in terms of training data. The training data contain both normal and anomalous data. In its purest form, unsupervised anomaly-detection schemes use unlabeled data for training. Eskin *et al.* provided a geometric framework to realize an unsupervised anomaly-detection scheme [14]. In their work, raw input data are mapped to a feature space. They labeled points that are in sparse regions of the feature space as anomalies. They used three different algorithms: a fixed-width clustering-based algorithm, an optimized *K*-nearest-neighbor algorithm, and an unsupervised variant of the SVM algorithm. Two assumptions are made about the data. The first assumption is that the number of normal instances far outnumbers the anomalous instances. The second assumption is that the anomalies are qualitatively different from the normal instances. Both these assumptions are not necessarily valid in all cases. For instance, assumption 1 will not hold for detection of certain denial of service (DOS) attacks. Oldmeadow *et al.* did further work on the clustering method used in [14] and obtained an improvement in accuracy when the clusters are adaptive to changing traffic patterns [19]. Leung and Leckie achieved a more computationally efficient algorithm based on hybrid clustering techniques such as clustering in Quest (CLIQUE) and parallelized merging of adaptive finite intervals (pMAFIA) [20]. The hybrid clustering methods they employed combine the ideas of grid-based and density-based clustering. They evaluated the algorithm using the KDD Cup 1999 data set after filtering out much of the attack records so that the assumption that majority of the network connections are normal traffic is valid for both training and test data. Though it is true that unsupervised anomaly-detection schemes, as described in [14], [19], and [20], do not need labeled training data, they also have the restriction that the percentage of attacks should be much less compared to that of normal instances. Besides this, the data records, when mapped to a feature space, does not give enough separation of normal records and attack instances, thus resulting in false identification of attacks as normal and vice versa. Other published results on anomaly-detection schemes for IDS include those that use various data-mining techniques such as clustering, variations of SVMs, and neural-network models [11], [16], [17], [22], [26], and [32]. A cost-efficient implementation of a multilayer hierarchical Kohonen network is presented in [13]. The above paper also gives a more comprehensive survey of intrusion-detection schemes.

The specification-based approach of intrusion detection is relatively new. Like anomaly detection, specification-based techniques detect attacks as deviants from a norm. The difference is that the specification-based techniques depend on manually developed specifications that capture legitimate system behavior. No machine-learning techniques are used here. Such systems are said to avoid the high false-alarm rates caused by legitimate but unseen behavior [23]. However, specification-based techniques need a considerable effort in the development of detailed specifications. Insufficient and poorly defined specifications can lead to a low detection rate.

In this paper, we concentrate on the anomaly-based approach for a system of feature-based intrusion detection. The development of the approach is based on a systematical analysis of higher order modeling functions of the input data sets. It is known that one drawback of most anomaly-based approaches is the problem of accurately classifying data sets that consist of a number of highly intertwining (mixed) data points of different categorical nature in a multidimensional space. Such inaccurate classification leads to a reduced detection rate and/or increased false-positive rate. Sarasamma *et al.* addressed this problem to a certain extent by using a high-order nonlinear classifier model [13]. The approach creates clusters that model the intersection of hypercylinders in a computationally efficient way.

Anomaly detection in network packet header data can be formulated as a computational-model reconstruction process. To be more precise, we use nonlinear functions to model network-connection records by applying a geometric-reconstruction process that reveals the underlying properties of the data set. Geometric reconstruction has been popularly applied in engineering as a process of discovering the original structure that generated the data set. The K-means clustering is a typical example of geometric reconstruction, which can be stated as follows: Given a set of data points in R^d space, find a set of k functions that best approximate the underlying functions that produced the data. Partitioning a set of data objects into homogeneous groups or clusters with certain intrarelations is a fundamental operation in geometric reconstruction. In K-means clustering, the functions to be reconstructed are formulated in terms of a set of parameters representing the centers and diameters of K isometric geometries—the clusters. When Euclidean distance is used as the metric, the geometries are hyperspheres. If the metric used is Mahalanobis distance, the geometries are hyperellipsoids.

We present a new approach for building ellipsoidal clusters for multidimensional data. The novelty of the technique lies in the fact that the parameters needed to construct general multivariate Gaussian functions of the higher order data are incrementally derived using accretive functions. We use a feedforward neural network that has an input layer, a single

competitive layer, and an output layer to achieve this. A radial basis function is used as the generator. An evaluation based on the inclusiveness and exclusiveness of samples with respect to specific criteria is used to accretively build the output clusters of the neural network. We apply this technique to realize an anomaly-detection system for network-based intrusion detection. This anomaly-detection system uses unsupervised learning. The training data are not restricted to just normal data. First, we use feature vectors extracted from the KDD Cup 1999 data set as the training and testing data. Detailed description of the KDD Cup 1999 data can be found in [2], [3], [11], and [12]. The goal is to detect as many different attack types as possible. Next, we use preprocessed network dump data to evaluate the capability of the system.

The rest of this paper is organized as follows. In Section II-A, we first present a more formal description of the accretive building process of the hyperellipsoidal clusters. We present the training algorithm and detection algorithm for a neural-network-based realization of this computational model in Section II-B. Section II-C covers some implementation aspects and discusses test data and results for the KDD Cup 1999 data. In Section II-D, we compare the results of the ellipsoidal SOM (ESOM) with that of other neural-network-based classifiers such as adaptive resonance theory (ART), radial basis function (RBF), and hierarchical K-Map. Results for tests conducted on actual network data that were processed using tcptrace are presented in Section II-E. Finally, in Section III, we discuss related work and conclude with our results.

II. Anomaly Detection Using Min–Max Hyperellipsoidal Clusters

A. Clustering by Geometric Reconstruction

Geometric reconstruction deals with the algorithmic problems of combining the results of one or more measurements of some aspect of a physical or mathematical object to obtain certain desired information about the object [30]. A simple form of reconstruction is static reconstruction, where we consider inverse problems of the following type. Let U be a geometric structure and T a transformation such that $T(U) \rightarrow V$, where V is a different geometric structure. Now, given T and V, construct a structure U' such that $T(U') \rightarrow V$. If T is a one-to-one mapping, then $U = U'$. The essence of geometric reconstruction is that of modeling a data set by single or multiple linear or nonlinear function(s) that reflect or reveal the underlying properties of the data collection. In other words, it is the process of discovering the original geometric structure that generated the data set. A typical example is the K-means clustering problem. It is well known that function reconstruction in general is an ill-posed inverse problem [5] since:

1) existence criterion may be violated because a distinct output function may not exist for every input data point;
2) uniqueness criterion may be violated because there might be multiple possible mappings from the data points to the function sets;
3) continuity (stability) criterion may be violated because of the unavoidable presence of noise or imprecision in the data set introduced in the direct mapping process (in the process of generating the data set from the physical world).

Knowing that geometric-reconstruction problems are ill posed, the next question is, under what conditions are the resulting functions exact or close to the exact solutions with respect to the physical world's originality. Often, the existence criterion can be forced by enlarging or reducing the solution space (the space of "models") and by formulating the objective functions in positive and symmetrical, though possibly discontinuous, distributions [30]. Questions on the solutions for uniqueness and stability still remain. With additional *a priori* information on the solutions such as smoothness and bounds on the derivatives, it is possible to restore stability and construct efficient numerical algorithms for approximate solutions with sufficient accuracy [5]. These methods would follow the fundamental regularization theory established in the 1960s [4]. A comprehensive review of regularization theory can be found in [36]. The basic idea of regularization is to stabilize the solution set by means of some auxiliary nonnegative function that embeds prior information about the solution. The most common form of prior information involves the assumption that the input–output mapping function (solution to the reconstruction problem) is smooth, in the sense that similar inputs correspond to similar outputs.

In our paper, we apply geometric reconstruction to do discriminatory data analysis. Here, we deal with the relations between a set of known classes denoted as $\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\}$ and a set of known data points in an n-dimensional space denoted as $\mathbf{x} = [\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n]$. The total possible occurrences of these points form the n-dimensional space $R(\mathbf{x})$. Collections of $\mathbf{x}$ partition $R(\mathbf{x})$ into regions $R(\omega_i) i = 1, 2, \ldots, c$; where $R(\omega_i) \subseteq R(\mathbf{x}), \cup R(\omega_i) = R(\mathbf{x})$, and $R(\omega_j) \cap R(\omega_i) = \emptyset,\ \forall j \neq i$. The $R(\omega_i)$ represent clusters of $\mathbf{x}$ based on the characteristics of class ω_i. The surfaces known as decision boundaries that separate these $R(\omega_i)$ regions are described by discriminant functions denoted as $\pi_i(\mathbf{x})$. This formulation can also be described as $R(\omega_i) = \{\mathbf{x} | \forall j \neq i, \pi_i(\mathbf{x}) > \pi_j(\mathbf{x})\}$. Often, the $R(\omega_i)$ regions are convex and continual, which renders $\pi_i(\mathbf{x})$ to be linear or piecewise-linear functions [25]. However, there are cases where these functions are nonlinear due to irregular and complex distributions of the feature vectors. Methods of applying linear or piecewise-linear approximations tend to have less statistical precision embedded in the pattern class distributions. That is, to form precise $R(\omega_i)$ regions, the $\pi_i(\mathbf{x})$ functions are required to be in high-order nonlinear forms. Such functions, if not totally impossible, are computationally expensive to realize. Methods based on machine learning and fuzzy clustering help the model adapt to nonlinearity, but they lack precision due to the restriction of data classes to a single distribution function. Separate efforts by Avi-Itzhak *et al.* [1], Kudo [6], and Zhu *et al.* [30] modeled complexly distributed data sets as a number of subsets, each with a relatively simple distribution. In this modeling, subset regions are constructed within a multidimensional data space. Data collections in such subspaces have high intrasubclass and low intersubclass similarities. The overall distribution of the data class is a combining set of the distributions of the subclasses (not necessarily additive). In that sense,

subclasses of one data class are the component clusters of the data set.

To facilitate the understanding, we represent the fundamentals of the technique from [30] as follows. Let S be a set of data points $\mathbf{x}_k$ in which each data point $\mathbf{x}_k$ is associated with a specific class S_j; that is, $S = \bigcup_{i=1}^{c} S_i, S_i \cap S_j = \emptyset$; $\forall i \neq j$, where S_i is a set of data points that are labeled by ω_i, $\omega_i \in \Omega = \{\omega_i; i = 1, 2, \ldots, c\}$. That is, for each $\mathbf{x}_k \in S$, there exists an i such that $[(\mathbf{x}_k \in S_i) \Rightarrow (\mathbf{x}_k \in \omega_i)]$.

Definition 1: Let S_i be a set of data points of type (category) ω_i, $S_i \subseteq S$, and $\omega_i \in \Omega$. Let ε_{ik} be the kth subset of S_i. That is, $\varepsilon_{ik} \subseteq S_i$, where $k = 1, 2, \ldots d_i$, and d_i is the number of subsets in S_i. Let $P(\mathbf{x}|\varepsilon_{ik})$ be a distribution function of the data point $\mathbf{x}$ included in ε_{ik}. The subclass clusters of S_i are defined as the set $\{\varepsilon_{ik}\}$ that satisfies the following conditions:

1) $\bigcup_{k=1}^{d_i} \varepsilon_{ik} = S_i$;
2) $\varepsilon_{ik} \cap \varepsilon_{il} = \emptyset, \forall l \neq k$;
3) $(\mathbf{x} \in \varepsilon_{ik}) \Rightarrow P(\mathbf{x}|\varepsilon_{ik}) > P(\mathbf{x}|\varepsilon_{il}), \forall l \neq k$;
4) $(\mathbf{x} \in \varepsilon_{ik}) \Rightarrow P(\mathbf{x}|\varepsilon_{ik}) \geq P(\mathbf{x}|\varepsilon_{jl}), \forall j \neq i$

where $P(\mathbf{x}|\varepsilon_{jl})$ is a distribution function of the lth subclass cluster for the data points in category set S_j, i.e., data points of class ω_j. In the above definition, condition 3) describes the intraclass property and condition 4) describes the interclass property of the subclasses. Condition 4) is logically equivalent to $(\mathbf{x} \in \varepsilon_{jl}) \Rightarrow P(\mathbf{x}|\varepsilon_{jl}) > P(\mathbf{x}|\varepsilon_{ik}), \forall j \neq i$, where $P(\mathbf{x}|\varepsilon_{ik})$ is the distribution function of the kth subclass cluster for the data points in category S_i. Note that the above definition does not exclude a trivial case where each ε_{ik} contains only one data point. It is known that a classifier built on this case degenerates to a classical one-nearest-neighbor classifier. However, considering the efficiency of the classifier to be built, it is more desirable to divide S_i into a minimum number of subclass clusters. This leads to the introduction of the following definition.

Definition 2: Let ε_{ik} and ε_{il} be two subclass clusters of the data points in S_i, $k \neq l$, and $\varepsilon_{il} \neq \emptyset$. Let $\varepsilon_i = \varepsilon_{ik} \cap \varepsilon_{il}$ and $P(\mathbf{x}|\varepsilon_i)$ be the distribution function defined on ε_i. The set $\{\varepsilon_{ik}; k = 1, 2, \ldots, d_i\}$ is a minimum-set subclass-cluster set of S_i if, for any $\varepsilon_i = \varepsilon_{ik} \cup \varepsilon_{il}$, we would have

$$\exists (j \neq i) \exists (\mathbf{x} \in \varepsilon_{jm}) \left[P(\mathbf{x}|\varepsilon_i) > P(\mathbf{x}|\varepsilon_{jm})\right]$$

or

$$\exists (j \neq i) \exists (\mathbf{x} \in \varepsilon_i) \left[P(\mathbf{x}|\varepsilon_i) < P(\mathbf{x}|\varepsilon_{jm})\right].$$

The above definition means that every subclass cluster must be large enough such that any joint set of them would then violate the subclass definition [condition 4)]. According to condition 3) of the subclass definition, a subclass region $R(\omega_{ik})$ corresponding to the subclass ε_{ik} can be defined as $R(\omega_{ik}) = \{\mathbf{x}|P(\mathbf{x}|\varepsilon_{ik}) > P(\mathbf{x}|\varepsilon_{il}), \forall l \neq k\}$. Thus, $P(\mathbf{x}|\varepsilon_{ik})$ can be viewed as a distribution function defined on the feature vectors $\mathbf{x}$ in $R(\omega_{ik})$. Combining this with condition 2) of the subclass-cluster definition, we have

$$R(\omega_{ik}) \cap R(\omega_{il}) = \emptyset, \qquad \forall l \neq k$$
$$R(\omega_{ik}) \cap R(\omega_{jl}) = \emptyset, \qquad \forall j \neq i.$$

The subclass clusters, thus, can be viewed as partitions of the decision region $R(\omega_i)$ into a number of subregions $R(\omega_{ik}), k = 1, 2, \ldots, d_i$, such that $R(\omega_{ik}) \subseteq R(\omega_i)$, and $\bigcup_k R(\omega_{ik}) = R(\omega_i)$.

Observing the fact that $R(\omega_{ik}) \cap R(\omega_{jl}) = \emptyset, \forall j \neq i$, we have $R(\omega_i) \cap R(\omega_j) = \emptyset, \forall j \neq i$. When a multivariate Gaussian distribution function is assumed for the data distribution, the probability that a data point $\mathbf{x}$ lies within a cluster region ω_i is given by

$$P(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_i|^{\frac{1}{2}}} e^{\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^{\mathrm{T}} \Sigma_i^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)\right]}.$$

Geometrically, samples drawn from a Gaussian population tend to fall in a single cluster region. In this cluster, the mean vector $\boldsymbol{\mu}_i$ determines the center of the cluster and the covariance matrix $\boldsymbol{\sigma}_i$ determines the shape of the region and $\mathbf{x}$, a column vector, represents the input data point. The locus of points of constant density for a multivariate Gaussian distribution forms a hyperellipsoid in which the quadratic form $r = (\mathbf{x} - \boldsymbol{\mu}_i)^{\mathrm{T}} \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)$ is equal to a constant. The eigen vectors of $\boldsymbol{\sigma}_i$ give the principal axes of the hyperellipsoid and the eigen values determine the lengths of these axes. The scalar $[(\mathbf{x} - \boldsymbol{\mu}_i)^{\mathrm{T}} \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)]^{1/2}$ is the Mahalanobis distance. Thus, the contour of constant density of a multivariate Gaussian distribution is a hyperellipsoid with a constant Mahalanobis distance to the mean vector $\boldsymbol{\mu}_i$. The volume of the hyperellipsoid measures the scatter of the samples around the point $\boldsymbol{\mu}_i$ [7]. Given a set of data samples of class ω_i in a multivariate Gaussian distribution, the determination of the function $P(\mathbf{x}|\omega_i)$ can be viewed approximately as a process of clustering the samples into a hyperellipsoidal subspace described by $r \leq C, C > 0$. The value of C is a constant that determines the scale of the hyperellipsoid. The parameter C should be chosen such that hyperellipsoids properly cover the data points in the set. This leads us to the topic of min–max hyperellipsoidal data characterization.

B. Min–Max Hyperellipsoidal Clustering

We apply an accretive approach to characterize the hyperellipsoidal clusters and use an artificial neural network (ANN) model to incrementally learn the parameters μ and covariance Σ of the data samples that are to be classified. This ANN has an input layer, a single competitive layer, and an output layer (see Fig. 1 for a graphical illustration). The data samples are presented to the input layer. The output layer represents the classification of hyperellipsoidal clusters. We use multivariate Gaussian function as the input–output (transfer) function at the middle layer of the neural network. The uniqueness of our approach is in the fact that the parameters for the transfer function are not predetermined by examining the data samples. Instead, we use an accretion approach to process each data sample one by one through the following steps.

1) Find the hyperellipsoid whose center has the shortest distance from the data sample. Call it the winner.
2) Merge this sample with that hyperellipsoid if the smallest hyperellipsoid that contains the samples is the winner and

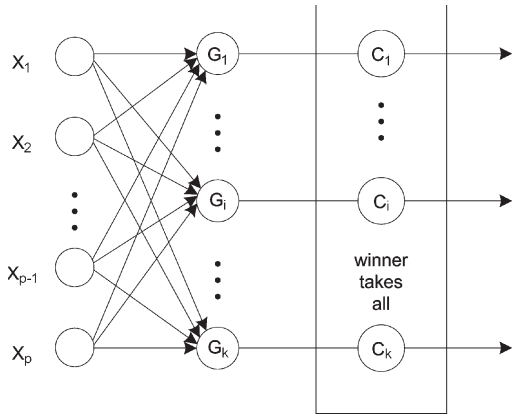

Fig. 1. ANN for the hyperellipsoidal SOM, where x_i represents custom column vectors, nodes marked by letter G_i are neurons with Gaussian radial basis function, and C_i are output clusters resulting from a winner-take-all strategy.

the new sample does not intersect with any other existing hyperellipsoid. Update μ and Σ.

3) If the merger in step 2) is not possible, then create a new hyperellipsoid with the data vector as its mean μ and the identity matrix as its Σ.

We use the winner-take-all SOM shown in Fig. 1 to generate min–max hyperellipsoidal clusters with high intracluster and low intercluster similarities (please refer to [24] for a detailed description of SOMs). Feature vectors extracted from network-connection records form the input data space. Each feature vector is passed to the column of neurons in the input layer. The transfer function G is applied on the input vector. The competitive layer builds the hyperellipsoidal clusters based on the mean and covariance of the data vectors seen so far. A single neuron is selected as the winner and the data vector is mapped to the output cluster corresponding to the winner neuron. The winner is chosen as the neuron that has the least Mahalanobis distance between its mean and the data vector. To reduce the computational cost, we use the square of the Mahalanobis distance as the metric. The goal is to train the neural network with data that contain normal as well as attack records and then use the trained neural network as the min–max ellipsoidal classifier for new data. We utilize the classification capability of this ESOM to detect new anomalies in network header data. In the training phase, as each new data vector is fed to the input layer, the hidden layer updates the necessary parameters such as number of samples mapped to the winner neuron, the winner's current moving mean and current moving covariance. Note that the classifier architecture is that of the classical "minimum Mahalanobis distance" classifier. The novelty lies in the incremental procedure applied to compute the parameters of the Mahalanobis distance.

We use an unsupervised learning algorithm for training the hyperellipsoidal SOM. The training algorithm is given in the pseudocode algorithm TRAIN_ELLIPSOIDAL_SOM. First, we define some symbols and variables that are used in the algorithm. The column vector $\mathbf{x}$ represents an input feature vector. The mean of the data vectors mapped to the cluster associated with neuron i is denoted as $\boldsymbol{\mu}_i$. $\boldsymbol{K}_i$ denotes the covariance matrix for neuron i. K_i^{-1} denotes the inverse of the covariance matrix for neuron i. $G_i(\mathbf{x})$ denotes the Gaussian transfer function for neuron i. The minimum threshold that determines whether a data sample is within the ellipsoidal cluster corresponding to neuron i is identified as $t_{\min}$. $\mathbf{I}$ represents the identity matrix. In the context of this algorithm, N denotes the initial number of neurons. The number of neurons at any stage of training is represented by the variable $numNeurons$. Let n_i denote the number of data samples clustered on to neuron i. Let the variable $loopCount$ denote the current iteration. We will stop the training iterations when the number of changes in neuron mapping from cluster to cluster is within a preset value. This preset value is computed as the product of a threshold identified as $changeThreshold$ and the number of training samples $(numVectors)$. In rare cases, the $\boldsymbol{K}_i$ resulting from the merger step 2) may not have an inverse, though we did not observe such a case in the experiments that we tracked. To avoid any potential program crash in such cases, we added a provision to use the identity matrix instead of the inverse in such cases. In some cases, the stabilization may take a large number of iterations. To restrict such overly long training sessions, we also use a parameter known as $maxIterations$ to end the training. The variable $numChanges$ denotes the number of changes in neuron mapping within program iterations.

Algorithm TRAIN_ELLIPSOIDAL_SOM

Step 1: Get the necessary inputs from the user.
1 Number of initial neurons, N
2 Size of an input vector, p
3 Feature subset to be used
4 Minimum threshold to spawn a new neuron, $t_{\min}$
5 Threshold to determine the number of training iterations, $changeThreshold$
6 Maximum number of training iterations allowed, $maxIterations$
7 Names of input and state files

Step 2: Set up the initial N neurons.
$numNeurons \leftarrow N$;
for $i = 1$ to N, do
set $\boldsymbol{\mu}_i$ as a randomly selected data vector
$K_i \leftarrow \mathbf{I}; n_i \leftarrow 1$
end for

Step 3: Initialize iteration control parameters
$maxChanges \leftarrow changeThreshold * numVectors$
$loopCount \leftarrow 0$

Step 4: **repeat**
$loopCount \leftarrow loopCount + 1$;
$numChanges \leftarrow 0$
for each training vector $\mathbf{x}$, do
choose the winner as neuron j such that $[\mathbf{x} - \boldsymbol{\mu}_\mathbf{j}]^{\mathrm{T}} \mathbf{K}_\mathbf{j}^{-1} [\mathbf{x} - \boldsymbol{\mu}_\mathbf{j}]$ is the least
compute $G_j(\mathbf{x})$, $G_j(\mathbf{x}) \leftarrow e^{-1/2[(\mathbf{x}-\mu_\mathbf{j})^{\mathrm{T}} \mathbf{K}_j^{-1} (\mathbf{x}-\mu_\mathbf{j})]}$
if $(G_j(\mathbf{x}) < t_{\min})$, then
create a new neuron u
$numNeurons \leftarrow numNeurons + 1$
$\boldsymbol{\mu}_\mathbf{u} \leftarrow \mathbf{x}; \mathbf{K}_u \leftarrow \mathbf{I}; n_u \leftarrow 1$
else
find the neuron k on which $\mathbf{x}$ was previously clustered

if $(j \neq k)$, **then**

$n_j \leftarrow n_j + 1;$

$\boldsymbol{\mu}_\mathbf{j} \leftarrow \frac{n_j - 1}{n_j}\boldsymbol{\mu}_j + \frac{1}{n_j}\mathbf{x};$

$\mathbf{K}_\mathbf{j} \leftarrow \frac{n_j - 1}{n_j}\mathbf{K}_\mathbf{j} + \frac{1}{n_j}[\mathbf{x} - \boldsymbol{\mu}_\mathbf{j}][\mathbf{x} - \boldsymbol{\mu}_j]^{\mathrm{T}};$

$n_k \leftarrow n_\mathrm{k} - 1;$

$\boldsymbol{\mu}_\mathbf{k} \leftarrow \frac{n_k + 1}{n_k}\boldsymbol{\mu}_\mathbf{k} - \frac{1}{n_k}\mathbf{x};$

$\mathbf{K}_k \leftarrow \frac{n_k + 1}{n_k}\mathbf{K}_k - \frac{1}{n_k}[\mathbf{x} - \boldsymbol{\mu}_k][\mathbf{x} - \boldsymbol{\mu}_k]^{\mathrm{T}}$

$numChanges \leftarrow numChanges + 1$

end if

end else

end for

until $(numChanges < maxChanges)$ or $(maxIterations < loopCount)$

The trained ESOM could be used as a pure anomaly detector or as a classifier. Currently, our goal is to apply the excellent classification capability of the hyperellipsoidal SOM in detecting as many different types of attacks as possible. Therefore, once the unsupervised training is completed, we label the clusters thus created. We examine the composition of each ellipsoidal cluster created. Records in some clusters will correspond to a unique label. However, there will be clusters that contain records that correspond to different labels. We use the term homogeneous clusters to denote the clusters of a specific label. The homogeneous clusters are labeled with the specific label of its member records. For the nonhomogeneous clusters, we apply the following labeling strategy.

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

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

Let $A = \{a_1, a_2, \ldots, a_\mathrm{r}\}$ 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 to indicate the level of confidence with which we choose the type of record that dominates the cluster. The confidence factor of cluster i is computed as

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

These steps of labeling and computing confidence can be formalized as follows.

Algorithm Finalize Training

for each cluster i, **do**

if cluster i is homogeneous

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

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

else begin

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

for each label X in cluster i, compute

$$F_i(x) = \left(\frac{X_i}{X_{\mathrm{total}}}\right)\left(\frac{X_i}{N_i}\right)$$

Set the label for cluster i, $L(i) \leftarrow \alpha$, where $F_i(\alpha) = \max_{\mathrm{x} \in A} F_i(x)$.

Set confidence factor

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

end else

end for

The trained state of the neurons is stored in files called state files. In addition to the parameters such as $\boldsymbol{\mu}$ and $\mathbf{K^{-1}}$, we also save the label and confidence of each neuron. The detection phase consists of loading a trained state in the ESOM and feeding each feature vector corresponding to the test set to the input layer. Now, to improve the computational efficiency of the detection phase, we computed the inverses of the covariances after the training and stored the $\mathbf{K}_i^{-1}$ in the trained-state file. The neuron that yields the smallest Mahalanobis distance between the test data vector and the mean of the samples clustered to that neuron is selected as the winner. The detection algorithm may be formalized as follows.

Algorithm Detect Events

Step 1: Initialize the ESOM with the parameters stored in a state file. Let C denote the total number of clusters in the state file.

Step 2: **for** each test record r, **do**

Construct the feature vectors using the user-supplied feature subspace.

for each cluster $i = 1$ to C, **do**

Compute $Distance(i) \leftarrow [\mathbf{x} - \boldsymbol{\mu}_i]^{\mathrm{T}}\mathbf{K}_i^{-1}[\mathbf{x} - \boldsymbol{\mu}_i]$.

end for

$winner(r) \leftarrow \min_i Distance(i)$

end for

Step 3: Compute the false positives and detected anomalies.

C. Implementation Aspects of Min–Max Hyperellipsoidal SOM

From our experiments in earlier studies [13], it became clear that feature vectors that include every single feature of a network-connection record need not give the best classification of the feature space. Furthermore, usage of large feature

TABLE I
COMPOSITION OF KDD-CUP-1999-BASED TRAINING AND TEST SETS

Type	KDD99 10%	Training Set 1	Training Set 2	Test set 1	Test set 2
normal	97,278	1129	97,278	1102	60,593
neptune	107,201	2	45,700	2	58,001
smurf	280,790	168	17,270	91	164,091
back	2,203	44	2,203	37	1,098
satan	1,589	2	1,589	0	1,633
ipsweep	1,247	14	1,247	14	306
portsweep	1,040	39	1,040	17	354
warezclient	1,020	0	1,020	0	0
teardrop	979	0	979	0	12
pod	264	0	264	0	87
nmap	231	0	231	0	84
Guess_passwd	53	38	53	38	4,367
Buffer_overflow	30	0	30	2	22
Land	21	0	21	0	9
warezmaster	20	0	20	0	1,602
imap	12	0	12	0	1
rootkit	10	0	10	0	13
loadmodule	9	1	9	1	2
ftp_write	8	2	8	0	3
multihop	7	0	7	0	18
phf	4	1	4	0	2
perl	3	1	3	1	2
spy	2	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	1441	169,000	1305	311,029

vectors increases the computational cost. Attacks that exploit the flaws and weaknesses in certain aspects of a protocol tend to leave their mark on a small subset of specific features of the network-connection record. One of our goals is to identify the specific subsets of features that are capable of detecting each attack category. Another goal is to find the subset (s) of features that detects the maximum number of attacks (both previously seen and novel) at the least possible false-positive rate. We chose the KDD 1999 data set as our first set of data for training and testing experiments. We chose this data set for four reasons. First, it has been used popularly as a standard for comparing the performance of intrusion-detection systems. Hence, we can judge the performance of the hyperellipsoidal clustering technique. Second, since the data is labeled, we can verify the accuracy of our detection scheme. Third, the test set contains 17 additional attack types that are not present in the training data. Lastly, the test data and training data set have no common elements. In the context of the KDD Cup 1999 data, each connection record encapsulates the information such as basic traffic features, features derived from observations in the past 100 connections, features derived from observations in the past 2 s, and content features. The KDD Cup 1999 data are organized in files as connection records in American standard code for information interchange (ASCII) format. Each connection record consists of a comma-delimited set of 41 features and a label that indicates whether the record is normal or an attack. To measure the effects of feature subsets and the minimum threshold for spawning new clusters, we designed a menu-driven interactive graphical user interface.

The metrics for evaluating the performance of the algorithms are as follows. If a data record labeled as normal is detected as an anomaly, then we consider it as one instance of false positive. If an anomaly record is identified as normal, then we consider it as a case of missed detection. Let N_{false} denote the total number of false positives and N_{missed} denote

TABLE II
COMPOSITION OF FEATURE SUBSETS

Set	Features	Set	Features	Set	Features
1	SrcBytes DstBytes Service Flag Protocol DstHstSameSrcPortRate Hot LogedStatus DstHstSvcDiffHstRate	7	LandStatus SU attempts NumFileCreation Failed Logins SameHstSameSvcRate NumShells NumRootAccess DstHstSameSrcPortRate	14	Hot Failed Logins NumCompromised NumRootAccess NumFileCreation NumShells NumAccessFiles NumOutBoundCommands
2	SameSvcSynErrRate SameHstSameSvcRate SameHstDiffSvcRate SameSvcDiffHstRate DstHstSvcDiffHstRate DstHstSameSvcRate DstHstSameSrcPortRate Count sameHstSynErrRate	8	Hot Failed Logins NumFileCreation SameHstRejErrRate SameSvcSynErrRate SameHstSameSvcRate SameHstDiffSvcRate SameSvcDiffHstRate dstHstSvcDiffHstRate	15	Count SameHstSynErrRate SameHstRejErrRate SameHstSameSvcRate SameHstDiffSvcRate SameHstSameSvcCnt SameSvcSynErrRate SameSvcRejErrRate SameSvcDiffHstRate
3	DstBytes SrcBytes Protocol Service Wrong Fragments	9	Flag NumRootAccess LogedStatus DstHstSameSvcRate DstHstSameSrcPortRate Count SameHstSynErrRate	16	Flag Hot Failed Logins DstHstSameSvcRate DstHstSameSrcPortRate LogedStatus
4	DstBytes SrcBytes Protocol Service Count	10	DstBytes SrcBytes Protocol service	17	DstBytes SrcBytes Protocol Wrong Fragments
5	SameSvcSynErrRate SameHstSameSvcRate SameHstDiffSvcRate SameSvcDiffHstRate DstHstSvcDiffHstRate Hot Failed Logins NumFileCreation LogedStatus	11	Failed Logins NumFileCreation NumRootAccess SameHstRejErrRate SameSvcSynErrRate SameHstSameSvcRate SameHstDiffSvcRate SameSvcDiffHstRate Wrong Fragments	18	Duration Protocol Service Flag SrcBytes DstBytes LandStatus Wrong Fragments Urgent Packets
6	Service Flag NumRootAccess DstHstSameSvcRate DstHstSameSrcPortRate Count SameHstSynErrRate	12	Wrong Fragments Flag NumRootAccess LogedStatus Hot Failed Logins NumFileCreation	19	SameHstSynErrRate SameSvcSynErrRate SameHstRejErrRate SameSvcRejErrRate SameHstSameSvcRate SameHstDiffSvcRate SameSvcDiffHstRate
		13	SrcBytes DstBytes Flag Service SameSvcRejErrRate LogedStatus DstHstSvcCount DstHstSvcRejErrRate	20	GuestLoginStatus Flag Hot LogedStatus DstHstSvcDiffHstRate DstHstSameSvcRate DstHstSameSrcPortRate Count sameHstSynErrRate

the number of missed instances encountered in the detection phase. Let N_{normal} and N_{attack} denote, respectively, the number of normal records detected and the number of attack records detected in the test set. The percentage of false positives is computed as $\%\text{FalsePositive} = (N_{\text{false}}/N_{\text{normal}}) * 100$. The percentage of detected anomalies is computed as $\%\text{Detected} = (N_{\text{attack}} - N_{\text{missed}})/N_{\text{attack}} * 100$.

For the experiments on KDD Cup 1999 data, we used two subsets of the 10% labeled training data. Training set 1 consists of 1441 records and training set 2 consists of 169 000 records. Training set 2 contains all the attack types with an adequate representation of all the service types, flags, and protocol types. For the detection phase, we used the labeled data set identified as "corrected" in the KDD Cup 1999 suit as one test set and a smaller subset of the 10% training set as the second test set. The composition of the two training sets and the test sets are listed in Table I. For first-level training and testing, we used "training set 1" and "test set 1," respectively. To study the effect of feature subsets on the detection rate and false-positive rate, we created 20 feature subsets. The composition of these feature subsets is listed in Table II.

1) Test Results on KDD Cup 1999 Data Training Set 1 and Test Set 1: We tested our algorithm on the training set 1 and test set 1 of the KDD Cup 1999 data. For the first 16 test cases with respect to the feature subsets, we used a $t_{\min}$ threshold of 0.75. The results are listed in Tables III and IV. Table III lists the results for experiments with feature subsets labeled sets 1–5 and sets 8–11.

TABLE III
RESULTS OBTAINED BY APPLYING NINE FEATURE SUBSETS TO TRAINING SET 1 AND TEST SET 1

Event type	Set 1	Set 2	Set 3	Set 4	Set 5	Set 8	Set 9	Set 10	Set 11
normal.	95.92	83.67	95.37	90.56	95.55	68.78	85.30	95.37	99.82
buffer_overflow.	**0.00**	**100.00**	**0.00**	**0.00**	**100.00**	**100.00**	**50.00**	**0.00**	**100.00**
loadmodule.	0.00	100.00	0.00	0.00	100.00	100.00	100.00	0.00	100.00
perl.	0.00	100.00	0.00	0.00	100.00	100.00	100.00	0.00	100.00
neptune.	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
smurf.	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00
guess_passwd.	100.00	42.11	100.00	100.00	100.00	100.00	94.74	100.00	100.00
portsweep.	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
ipsweep.	92.86	100.00	92.86	92.86	92.86	92.86	100.00	92.86	92.86
back.	100.00	67.57	100.00	100.00	100.00	100.00	78.38	100.00	35.14
%Overall Detection	97.54	83.25	97.54	97.54	99.51	99.51	94.58	97.54	42.86
%FP	4.08	16.33	4.63	9.44	4.45	31.22	14.70	4.63	0.18

TABLE IV
RESULTS OBTAINED BY APPLYING DIFFERENT FEATURE SUBSETS TO TRAINING SET 1 AND TEST SET 1

Event type	Set 12	Set 13	Set 14	Set 16	Set 18	Set 20	ALL
normal.	0.18	100.00	97.82	90.93	95.46	96.28	91.20
buffer_overflow.	**100.00**	**100.00**	**100.00**	**100.00**	**50.00**	**100.00**	**50.00**
loadmodule.	100.00	100.00	100.00	100.00	0.00	100.00	100.00
perl.	100.00	100.00	100.00	100.00	0.00	100.00	100.00
neptune.	100.00	100.00	0.00	100.00	100.00	100.00	100.00
smurf.	100.00	100.00	0.00	100.00	100.00	100.00	98.90
guess_passwd.	100.00	100.00	100.00	100.00	100.00	100.00	100.00
portsweep.	100.00	100.00	0.00	100.00	100.00	100.00	100.00
ipsweep.	100.00	100.00	0.00	100.00	92.86	100.00	100.00
back.	100.00	100.00	100.00	100.00	100.00	94.59	100.00
%Overall Detection	100.00	99.01	38.92	100.00	98.03	99.01	99.01
%FP	99.82	5.26	2.18	9.07	4.54	3.72	8.80

Table IV summarizes the results for sets 12–14, 16, 18, 20, and lastly for the set that contains the entire 41 features. Each row with the exception of the last two rows in Tables III and IV gives the percentage of anomaly detection for a specific type of attack in the test data. Each column represents a subset of features used for training the ESOM. For instance, the first row in Table III indicates the percentage of normal records detected as normal by the hyperellipsoidal SOMs trained using the feature subsets listed as sets 1–5 and sets 8–11, respectively. The row identified as %Overall detection indicates the percentage of total anomalies detected in the test set when each feature subset is used. The last row gives the percentage of false positive for each feature subset. The test set contains two instances of an attack type, buffer_overflow, which is not present in the training set. It can be seen that sets 2, 5, 8, 11–14, 16, and 20 detected 100% of this new anomalous events. Sets 9, 18, and "ALL" detected 50% of the new anomaly. Sets 1, 3, 5, 10, 18, and 20 gave a detection rate above 97.5% at false-positive rates below 4.7%. The set "ALL" had a detection rate of 99.01% at a false-positive rate of 8.80%. Therefore, we observe that using the entire feature set to realize the ellipsoidal clustering, though computationally costly, does not yield the best clustering. We also note that though the overall detection rate for set 11 is only 42.86% at a false-positive rate of 0.18%, it detects 99.82% of the normal records as normal. Thus, employing a min–max hyperellipsoidal clustering based on the features identified in set 11 to filter out the normal records and then applying another ESOM trained using set 1 or 20 might yield better performance and good results. However, it should be noted that the training set and test set used for these 16 test cases are quite limited in scope. In the next suit of test cases, we use a more comprehensive set of training and test data.

2) Test Results for KDD Cup 1999 Data on Larger Training Set and Test Set: For the test cases in this section, we use training set 2, which is a more representative subset of the 10% KDD Cup 1999 training data. Besides the normal records, this training set contains 22 different attack types. Approximately 58% of records in this training set belong to the normal category. We use test set 2 for the detection phase in these test cases. The composition of the training set and test set are given in Table I. The results of 20 test cases are summarized in Tables V and VI. A $t_{\min}$ threshold of 0.45 was used for these test cases. At most, 100 training iterations were allowed. Table V contains the results for the first ten subsets of features and Table VI contains that of the remaining ten subsets. The overall percentage of detection is represented by the row labeled as "%Overall." The number of neurons generated and the percentage of false positives, respectively, are indicated in the last two rows. Each of the remaining rows shows the percentage detection of a specific event type in the test set. Anomaly event types not present in the training sets are highlighted in bold face. Note that all the 17 anomaly types not present in the training sets were detected, though by different feature sets with a different detection rate, in the test cases.

An examination of the results shows that 15 out of the 20 subsets yielded detection rates above 81%. Of these subsets, the results from sets 1, 16, 17, and 20 have false-positive rates below 7%. Subset 17 achieved a detection rate of 91.55% at a 2.68% false-positive rate and set 20 achieved a detection rate of 91.71% at a 4.84% false-positive rate. Subset 14 had the highest detection rate of 99.65% for normal records and, consequently, the least false-positive rate of 0.35%. However, the overall detection rate for subset 14 is extremely low for anomalous events. Another aspect that we looked at is the number of different attack types with above 80% detection rate for each feature subset. We also examined the number of new attack types with better than 80% detection rate for all subsets. These two results are summarized in Table VII. From Table VII, it can be seen that the feature subset 8 gives better than 80% detection rate for 32 different attack types. It can also

TABLE V
RESULTS FOR FEATURE SUBSETS 1–10 WITH TRAINING SET 2 AND TEST SET 2

Event type	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set 10
normal	96.92	82.70	89.56	86.17	75.21	88.98	51.20	60.50	86.60	89.75
snmpgetattack.	**0.00**	**41.39**	**0.10**	**20.51**	**99.97**	**37.17**	**89.26**	**99.51**	**0.03**	**0.10**
named.	**52.94**	**41.18**	**52.94**	**52.94**	**94.12**	**64.71**	**23.53**	**100.00**	**47.06**	**52.94**
xlock.	**0.00**	**100.00**	**22.22**	**33.33**	**77.78**	**22.22**	**44.44**	**100.00**	**66.67**	**11.11**
smurf.	99.99	99.97	100.00	100.00	99.98	81.78	99.95	99.98	100.00	100.00
ipsweep.	100.00	99.35	100.00	100.00	99.67	99.35	99.02	99.67	99.67	100.00
multihop.	44.44	66.67	38.89	33.33	94.44	44.44	66.67	88.89	72.22	22.22
xsnoop.	**0.00**	**100.00**	**0.00**	**0.00**	**100.00**	**75.00**	**75.00**	**100.00**	**100.00**	**0.00**
sendmail.	**17.65**	**88.24**	**47.06**	**5.88**	**47.06**	**82.35**	**41.18**	**82.35**	**64.71**	**35.29**
guess_passwd.	3.60	4.35	11.38	0.02	16.88	10.01	94.78	95.83	1.92	11.79
saint.	**98.10**	**97.01**	**98.51**	**99.05**	**96.47**	**98.10**	**97.83**	**98.10**	**98.10**	**99.05**
buffer_overflow.	13.64	77.27	4.55	27.27	59.09	50.00	86.36	77.27	68.18	4.55
portsweep.	98.87	74.01	100.00	73.73	95.76	90.40	66.67	100.00	67.51	100.00
pod.	100.00	98.85	100.00	100.00	91.95	90.80	86.21	89.66	95.40	100.00
apache2.	**34.51**	**32.37**	**34.51**	**35.26**	**28.21**	**76.70**	**100.00**	**79.22**	**32.75**	**34.51**
phf.	50.00	0.00	50.00	100.00	100.00	50.00	50.00	100.00	50.00	50.00
udpstorm.	**100.00**	**100.00**	**0.00**	**0.00**	**100.00**	**100.00**	**0.00**	**100.00**	**50.00**	**0.00**
warezmaster.	66.42	87.14	48.81	33.27	74.84	65.67	84.27	97.44	74.41	56.87
perl.	0.00	100.00	0.00	0.00	100.00	50.00	100.00	50.00	50.00	0.00
satan.	99.20	99.69	99.82	99.94	99.94	96.57	99.45	99.76	99.63	99.94
xterm.	**0.00**	**76.92**	**0.00**	**0.00**	**84.62**	**69.23**	**30.77**	**76.92**	**61.54**	**0.00**
mscan.	**93.64**	**51.85**	**94.21**	**91.45**	**25.36**	**83.19**	**42.36**	**66.38**	**83.57**	**93.73**
processtable.	**100.00**	**54.68**	**100.00**	**100.00**	**98.16**	**50.86**	**99.87**	**97.89**	**41.50**	**100.00**
ps.	**37.50**	**56.25**	**31.25**	**18.75**	**81.25**	**81.25**	**75.00**	**100.00**	**43.75**	**31.25**
nmap.	4.76	100.00	100.00	98.81	100.00	5.95	83.33	100.00	100.00	100.00
rootkit.	7.69	53.85	15.38	7.69	76.92	100.00	46.15	84.62	38.46	15.38
neptune.	71.40	99.88	100.00	99.98	99.98	99.97	99.72	100.00	99.98	100.00
loadmodule.	100.00	50.00	100.00	0.00	100.00	0.00	100.00	100.00	0.00	100.00
imap.	0.00	100.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	0.00
back.	95.81	68.76	95.81	95.81	99.36	1.09	100.00	99.91	79.42	95.81
httptunnel.	**84.18**	**83.54**	**89.24**	**87.34**	**96.84**	**94.30**	**77.85**	**96.84**	**89.87**	**90.51**
worm.	**0.00**	**100.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**100.00**	**0.00**	**0.00**
mailbomb.	**0.00**	**4.92**	**0.00**	**0.00**	**0.00**	**0.00**	**99.98**	**92.66**	**0.00**	**0.00**
ftp_write.	66.67	66.67	66.67	33.33	66.67	66.67	66.67	100.00	33.33	66.67
teardrop.	0.00	75.00	100.00	66.67	100.00	100.00	58.33	100.00	83.33	100.00
land.	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
sqlattack.	**0.00**	**100.00**	**0.00**	**0.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**0.00**
snmpguess.	**0.00**	**95.72**	**44.64**	**95.93**	**99.96**	**65.17**	**35.04**	**99.71**	**0.21**	**27.64**
Overall	85.07	93.68	92.21	92.98	95.79	81.17	98.40	99.50	91.44	92.10
#Neurons	239	596	156	398	337	353	76	354	295	145
%FP	3.08	17.30	10.44	13.83	24.79	11.02	48.80	39.50	13.40	10.25

be seen that this subset gives above 80% detection rate for 14 out of the 17 new attack types. For the remaining three attack types, mscan, xterm, and apache2 have the detection rates of 66.38%, 76.92%, and 79.22%, respectively. Attack types such as buffer_overflow, guess_passwd, and xsnoop had detection rates of 77.27%, 95.83%, and 100%, respectively.

D. Comparison of KDD Cup 99 Test Results With Other Algorithms

In this section, we compare the performance of ESOM with some of the other neural-network-based approaches such as multilayer perceptron (MLP), ART, RBF, hierarchical Kohonen network (HSOM), and other traditional algorithms such as maximum-likelihood Gaussian classifier (GAU), nearest cluster algorithm (NEA), K-means clustering algorithm (K-M), and hypersphere algorithm (HYP). For this purpose, we use the published results in [13], [26], and [32]. We compare the percentage detection rates and percentage false-positive rates for the basic four categories, namely PROBE, DOS, user-to-root (U2R), and remote-to-local (R2L), of attacks in Table VIII. The comparison is based on the test results we obtained for the KDD Cup 99 test set using instances of ESOM that were trained using training set 2 at a $t_{\min}$ threshold value of 0.45. The first column identifies the specific algorithm used. For the studies in [26] and [32], the effect of the feature subset is not considered. Judging the effect of feature subsets on the detection rate and false-positive rate is another goal in our paper and in [13]. Therefore, for ESOM and HSOM, we provide the results for several feature subsets. We have only a single value of %FP for each test case, whereas in [26], they have four different values of false-positive rates. Therefore, for the results of Sabhnani and Serpen, we list all four values for %FP. We also indicate the number of neurons used, where applicable.

TABLE VI
RESULTS FOR FEATURE SUBSETS 11–20 WITH TRAINING SET 2 AND TEST SET 2

Event type	Set 11	Set 12	Set 13	Set 14	Set 15	Set 16	Set 17	Set 18	Set 19	Set 20
normal	95.26	69.99	96.47	99.65	96.58	93.16	97.32	89.34	92.23	95.16
snmpgetattack.	**0.01**	**100.00**	**0.17**	**0.00**	**0.01**	**0.13**	**0.00**	**0.10**	**0.01**	**0.03**
named.	**23.53**	**100.00**	**88.24**	**11.76**	**23.53**	**76.47**	**29.41**	**47.06**	**23.53**	**41.18**
xlock.	**11.11**	**66.67**	**66.67**	**22.22**	**0.00**	**100.00**	**0.00**	**44.44**	**0.00**	**88.89**
smurf.	0.00	100.00	96.14	0.00	80.52	100.00	100.00	100.00	0.00	99.98
ipsweep.	15.69	100.00	96.08	0.00	15.36	100.00	100.00	100.00	15.69	99.67
multihop.	22.22	94.44	66.67	16.67	0.00	83.33	16.67	50.00	5.56	72.22
xsnoop.	**75.00**	**100.00**	**100.00**	**0.00**	**25.00**	**100.00**	**0.00**	**75.00**	**25.00**	**100.00**
sendmail.	**41.18**	**58.82**	**47.06**	**35.29**	**0.00**	**70.59**	**0.00**	**41.18**	**5.88**	**88.24**
guess_passwd.	16.81	0.82	1.76	0.00	1.12	16.35	5.15	3.48	1.37	4.47
saint.	**97.42**	**99.86**	**84.10**	**0.00**	**98.51**	**97.42**	**98.78**	**98.10**	**97.42**	**97.96**
buffer_overflow.	54.55	90.91	86.36	45.45	0.00	63.64	4.55	50.00	0.00	77.27
portsweep.	98.87	99.72	14.41	0.00	98.87	99.72	100.00	98.87	98.87	72.60
pod.	93.10	100.00	100.00	0.00	83.91	98.85	100.00	100.00	83.91	95.40
apache2.	**89.17**	**34.51**	**61.84**	**0.00**	**99.75**	**35.26**	**35.77**	**76.83**	**70.40**	**35.89**
phf.	50.00	100.00	100.00	50.00	50.00	50.00	50.00	50.00	50.00	0.00
udpstorm.	**0.00**	**100.00**	**100.00**	**0.00**	**0.00**	**100.00**	**100.00**	**100.00**	**0.00**	**50.00**
warezmaster.	0.94	99.88	83.96	36.08	0.81	99.81	33.21	66.42	1.37	93.57
perl.	50.00	50.00	100.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00
satan.	99.76	99.88	92.65	0.00	99.69	75.93	100.00	99.20	99.76	99.69
xterm.	**46.15**	**92.31**	**92.31**	**46.15**	**0.00**	**84.62**	**0.00**	**23.08**	**7.69**	**76.92**
mscan.	**32.57**	**86.89**	**85.38**	**0.00**	**24.22**	**62.39**	**90.88**	**93.73**	**76.64**	**80.72**
processtable.	**38.21**	**100.00**	**71.01**	**0.00**	**37.94**	**20.16**	**100.00**	**100.00**	**39.00**	**45.85**
ps.	**56.25**	**87.50**	**56.25**	**37.50**	**0.00**	**87.50**	**0.00**	**37.50**	**0.00**	**62.50**
nmap.	100.00	100.00	100.00	0.00	100.00	100.00	100.00	4.76	100.00	100.00
rootkit.	15.38	92.31	61.54	38.46	0.00	92.31	0.00	61.54	0.00	53.85
neptune.	99.98	100.00	50.55	0.00	100.00	99.93	100.00	71.82	100.00	99.98
loadmodule.	100.00	100.00	50.00	0.00	0.00	100.00	0.00	100.00	0.00	0.00
imap.	100.00	100.00	0.00	0.00	100.00	100.00	0.00	100.00	100.00	100.00
back.	10.38	99.36	95.81	99.36	10.11	99.45	95.81	95.81	11.29	98.72
httptunnel.	**86.71**	**100.00**	**97.47**	**1.27**	**86.71**	**93.04**	**84.18**	**84.81**	**86.71**	**93.04**
worm.	**0.00**	**0.00**	**100.00**	**0.00**	**0.00**	**100.00**	**0.00**	**0.00**	**0.00**	**100.00**
mailbomb.	**7.00**	**0.00**	**0.00**	**0.00**	**6.98**	**0.50**	**0.00**	**0.00**	**7.12**	**0.02**
ftp_write.	33.33	66.67	0.00	0.00	0.00	100.00	33.33	66.67	0.00	66.67
teardrop.	100.00	100.00	41.67	0.00	41.67	100.00	100.00	0.00	0.00	83.33
land.	100.00	100.00	100.00	0.00	88.89	100.00	100.00	100.00	100.00	100.00
sqlattack.	**100.00**	**100.00**	**100.00**	**0.00**	**0.00**	**100.00**	**0.00**	**0.00**	**0.00**	**100.00**
snmpguess.	**0.25**	**100.00**	**0.75**	**0.00**	**0.21**	**0.17**	**0.12**	**0.00**	**0.25**	**0.21**
Overall	25.43	96	77.64	0.68	77.89	91.70	91.55	85.32	25.27	91.71
#Neurons	224	33	656	45	278	189	116	260	236	426
%FP	4.74	30.01	3.53	0.35	3.42	6.84	2.68	10.66	7.77	4.84

E. Tests and Results for Network Dump Data

Here, we describe the tests conducted on actual network dump data collected in our own experimentation, and the results obtained for those tests. We used a closed network of computers to create the training and test data for the experiments. Different attack events were created using standard software tools. The packets were captured using Ethereal software and dumped into designated files. These network dump data were then processed using the tcptrace software. Tcptrace provides the capabilities to process the individual packets into connection records, where a connection record refers to all the packets that are transmitted from the initiation of a network connection to the graceful, or otherwise, termination of the connection. Besides the basic features of the connection such as number of bytes transmitted from the source to the destination and vice versa, service used, the protocol, the times, and so on, tcptrace also allows us to gather detailed statistics on each connection. At this time,

TABLE VII
NUMBER OF ATTACK EVENTS WITH ABOVE 80% DETECTION RATES

Feature Subset	All Attack Types	New Attack Types
Set 1	14	5
Set 2	20	9
Set 3	16	4
Set 4	15	5
Set 5	26	11
Set 6	18	7
Set 7	20	6
Set 8	32	14
Set 9	15	5
Set 10	16	4
Set 11	29	4
Set 12	14	12
Set 13	21	9
Set 14	2	0
Set 15	12	3
Set 16	25	9
Set 17	16	5
Set 18	16	5
Set 19	10	2
Set 20	20	8

TABLE VIII
COMPARISON OF ESOM RESULTS WITH OTHER QUALIFIERS

Algorithm/test case	% Detection Rate By Category				#Neurons used	%FP
ESOM	**Probe**	**DOS**	**U2R**	**R2L**		
set 1	95.73	90.18	59.46	1.24	239	3.08
set 2	84.93	97.27	76.83	39.37	596	17.30
set 3	98.20	97.24	61.78	11.04	156	10.44
set 4	95.39	97.13	59.85	26.89	398	13.83
set 5	80.10	97.39	89.96	74.99	337	24.79
set 6	91.31	83.97	84.56	33.78	353	2.68
set 7	81.59	99.77	72.59	81.72	76	48.80
set 8	91.05	99.73	92.28	98.42	354	39.50
set 9	92.58	97.13	76.45	0.90	295	13.40
set 10	98.22	97.30	61.39	8.34	145	10.25
set 12	96.59	97.61	96.53	70.23	33	30.01
set 13	83.05	82.35	87.64	1.08	656	3.53
set 15	74.15	82.85	52.90	0.45	278	3.42
set 16	80.58	97.34	88.42	5.38	189	6.84
set 17	97.48	97.14	52.90	1.65	116	2.68
set 18	95.75	90.44	70.66	1.33	260	10.66
set 20	92.29	97.37	82.24	1.69	426	4.84
HSOM						
comb4	87.88	97.19	66.52	0.37	144	2.92
comb3	88.43	99.44	66.95	0.40	144	3.99
comb2	88.24	97.17	71.24	20.89	144	3.00
comb1	67.95	73.96	67.81	20.46	144	3.35
Jalili and Amini – Best results						
ART-1	99.48	100	17.41	88.69	Output layer: 3000	2.24
ART-2	96.88	96.17	10.71	36.31	Output layer: 2000	1.06
Sabhnani and Serpen						
Multi-classifier	88.7	97.3	29.8	9.6		0.4, 0.4, 0.4,0.1
MLP	88.7	97.2	13.2	5.6		0.4, 0.3, 5E-2, 1E-2
GAU	90.2	82.4	22.8	9.6		11.3, 0.9, 0.5, 0.1
K-M	87.6	97.3	29.8	6.4		2.6, 0.4, 0.4, 0.1
RBF	93.2	73	6.1	5.9		18.8, 0.2, 4E-2, 0.3
NEA	88.8	97.1	2.2	3.4		0.5, 0.3, 6E-4, 1E-2
ART	77.2	97.0	6.1	3.7		0.2,0.3, 1E-3, 4E-3
HYP	84.8	97.2	8.3	1.0		0.4, 0.3, 9E-3, 5E-3

tcptrace can be used to process transmission control protocol (TCP) and user datagram protocol (UDP) records only. Internet control message protocol (ICMP) traffic embedded in IP packets cannot be parsed into connection records. More details on tcptrace can be found in [21].

For our training and test purposes, we created connection records by extracting a subset of the basic and detailed statistics derived by tcptrace for that connection. These features are listed in Table IX. Next, we selected subsets from the features thus extracted. These feature subsets were created by taking features pertaining to handshake in one group (handshake), features related to data transfer in another group (transfer), and general aspects of the connection in another group (general). Then, we created two subsets, one that combines feature sets general and handshake (GH), and the other that combines all three of the above subsets (GHT). The composition of these feature subsets is given in Table X. The test set and training set were disjoint. In addition to that, the test set contained anomaly events that were not present in the training set. The results are shown in Table XI. New anomalies not present in the training set are shown in boldface. The first column represents the event type. Each subsequent column indicates the results obtained for a specific combination of features used. The percentage of detection by event type is shown for each event type. The overall detection rate, the number of neurons generated, and the percentage of false positives are shown in the last three rows in that order. Again, in these test cases, our algorithm successfully detected seven new anomaly types that were not present in the training set.

An analysis of the results contained in Table XI shows that the overall detection rate is above 94% for each test case. The false-positive rate is below 5.7% for all test cases. The highest detection rate of 99.26% at a false-positive rate 5.68% was obtained for the subset general using 166 neurons. The feature subset GT gave a 98.69% detection rate at a false-positive rate of 2.32%. The transfer features yielded 98.32% detection for normal events, and hence, the best false-positive rate of 1.68%. The sets general and GH had above 80% detection for each of the 21 event types. GT is close behind with above 80% detection for 20 event types.

TABLE IX
FIELDS IN A CONNECTION RECORD

Field	Description	Data type
Source IP	IP address of the source	String
Source port	The port number of the source (also known as service)	Numeric
Destination IP	IP address of destination	String
Destination port	The port number of destination.	Numeric
Connection status	Did the connection complete gracefully or not? Was it reset?	String
First packet time	The date time stamp of the first packet seen in that connection. Format: Dow Mon DD HH:MM:SS.ssssss YYYY, Where Dow stands for day of the week, MON for the month, DD for date, HH for hour, MM for minute and SS.ssssss for the seconds with precision of the order of microseconds. YYYY represents the year.	String
Last packet time	Date time stamp of last packet. Format same as above.	String
Elapsed time	Duration of this connection in SS.ssssss format.	Numeric
numPktsSrcToDst	Total number of packets transmitted from source to destination.	Numeric
numPktsDstToSrc	Total number of packets from destination to source.	Numeric
numResetsSrcToDst	Number of resets sent from source to Destination.	Numeric
numResetsDstToSrc	Number of resets sent from destination to source.	Numeric
numAcksSrcToDst	Number of ack packets sent from source to destination.	Numeric
numAckDstToSrc	Number of ack packets sent from destination to source.	Numeric
uniqueBytesSrcToDst	Number of unique bytes sent from source to destination. Total bytes of data sent excluding retransmitted bytes and bytes sent doing window probing.	Numeric
uniqueBytesDstToSrc	Total unique bytes from destination to source excluding retransmitted bytes and window probing bytes.	Numeric
rexmtPktsSrcToDst	Number of retransmitted data packets from source to destination.	Numeric
rexmtPktsDstToSrc	Number of retransmitted data packets from destination to source.	Numeric
pushedPktsSrcToDst	Number of packets with the PUSH bit set in the TCP headers from source to destination.	Numeric
pushedPktsDstToSrc	Number of packets with the PUSH bit set in the TCP headers from destination to source.	Numeric
numSynPktsSrcToDst	Number of packets with SYN bits set in the TCP headers from source to destination.	Numeric
numFinPktsSrcToDst	Number of packets with SYN bits set in the TCP headers from destination to source.	Numeric
numSynPktsDstToSrc	Number of packets with FIN bits set in the TCP headers from source to destination.	Numeric
numFinPktsDstToSrc	Number of packets with FIN bits set in the TCP headers from destination to source.	Numeric
mssReqstSrcToDst	Maximum segment size requested in the SYN packet by the source.	Numeric
mssReqstDstToSrc	Maximum segment size requested in the SYN packet by the destination.	Numeric
initialWindowBytesSrcToDst	The total number of bytes sent in the initial window prior to receiving the first ack packet from the destination.	Numeric
initialWindowBytesDstToSrc	The total number of bytes sent in the initial window prior to receiving the first ack packet from the source.	Numeric

III. DISCUSSION AND CONCLUSION

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 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 1234-unit K-Map followed by a three-layer RPROP network of 70, 12, and 4 neurons, respectively. Sigmoid functions were used for each level of the RPROP network. The focus was only on three different attacks, the Neptune (an SYN flood attack), the satan probe, and the port-scan probe. They achieved a 90% detection rate for satan attacks at a 4.5% false-positive rate, a 97.9% detection rate for portsweep at a 4.19% false-positive rate, and a 99.72% detection rate for Neptune attacks at a 0.06% false-positive rate.

At Dalhousie University, Heywood and co-workers used SOMs 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 SOMs. The network-based IDS prototype used the preprocessed KDD Cup 99 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 99 Cup 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

TABLE X
FEATURE SUBSETS FOR EXPERIMENTS WITH NETWORK DUMP DATA

Subset	Features
General	ElapsedTime
	ConnectionStatus
	SrcPort
	DstPort
	InitialWindowBytesSrcToDst
	initialWindowBytesDstToSrc
HandShake	NumAckSrcToDst
	NumAckDstToSrc
	SynPktsSrcToDst
	FinPktsSrcToDst
	SynPktsDstToSrc
	FinPktsDstToSrc
Transfer	NumPktsSrcToDst
	NumPktsDstToSrc
	UniqueBytesSrcToDst
	UniqueBytesDstToSrc
	RexmtPktsSrcToDst
	RexmtPktsDstToSrc
	PushedPktsSrcToDst
	pushedPktsDstToSrc
GH	General plus Handshake
GT	General plus Transfer

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 an 89% detection rate at a false-positive rate of 4.6%.

Sarasamma *et al.* used a multilevel hierarchical Kohonen network to detect anomalous events in network data [13]. A cost-effective hierarchical extension of the simple Kohonen network was used to detect maximal number of attack types at low false-positive rates. They also evaluated the effects of various feature subsets on the detection rate and false-positive rate. Another motivation of the work in [13] was to achieve a high-order nonlinear classifier model to create clusters that model the intersection of hypercylinders in a computationally efficient way. Using a three-level hierarchical Kohonen net, they achieved detection rates between 90.94% and 93.46% at false-positive rates between 2.19% and 3.99% for three feature combinations. 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. When the attack types were limited to Neptune, satan, and portsweep, they achieved a 99.63% detection rate at a 0.34% false-positive rate. However, detection rates for attack types such as buffer_overflow, guess_passwd, and xsnoop in KDD Cup 1999 data were poor. One of the reasons for this was identified as the closely intertwined nature of the feature vectors of anomaly and normal events. Another reason was the fact that the feature vectors of some anomalous events closely matched the normal events.

In this paper, we create hyperellipsoidal clusters of maximum intra-cluster similarity and minimum intercluster similarity to more accurately classify data points of a highly intertwining nature, as seen in the KDD Cup 1999 data sets. We addressed this problem by accretively building hyperellipsoidal clusters at a slightly higher cost than that in [13]. We were able to get detection rates of 77.27%, 95.83%, and 100%, respectively, for the attack types buffer_overflow, guess_passwd, and xsnoop, which is a significant improvement over that in [13]. We achieved overall detection rates between 91.55% and 91.71% at false-positive rates between 2.68% and 4.84% when used for the entire attack range of the KDD Cup 1999 data. We

TABLE XI
RESULTS FOR NETWORK DUMP DATA

Event type	General	Handshake	Transfer	GH	GT
normal	94.32	97.19	98.32	95.09	97.68
cisco	100.00	100.00	90.91	100.00	100.00
dos	100.00	97.27	95.63	100.00	99.45
finger_abuses	100.00	100.00	66.67	100.00	100.00
firewalls	90.00	100.00	90.00	90.00	90.00
ftp	96.55	58.62	93.10	100.00	100.00
Gain_root	100.00	93.88	90.82	98.98	98.98
Gain_shell	100.00	100.00	97.50	100.00	100.00
General	97.89	96.84	85.26	96.84	96.84
port_scanner	100.00	100.00	99.95	99.64	100.00
Remote_file_access	97.73	97.73	93.18	100.00	100.00
smtp_problem	92.86	100.00	85.71	92.86	92.86
backdoor	97.73	97.73	96.59	97.73	97.73
SNMP	100.00	100.00	100.00	100.00	97.22
P2P_FileSharing	**100.00**	**94.74**	**68.42**	**94.74**	**94.74**
Misc	**95.74**	**95.04**	**86.52**	**92.91**	**97.16**
NIS	**100.00**	**100.00**	**75.00**	**100.00**	**100.00**
RPC	**83.33**	**100.00**	**66.67**	**83.33**	**83.33**
Unix_account	**100.00**	**100.00**	**96.92**	**100.00**	**100.00**
Useless_service	**88.89**	**88.89**	**37.78**	**80.00**	**91.11**
windows	**97.87**	**35.11**	**22.34**	**90.43**	**79.79**
%overall	99.26	96.54	94.05	98.49	98.69
#neurons	166	90	82	171	261
%FP	5.68	2.81	1.68	4.91	2.32

were able to obtain above 80% detection for 32 of the different attack events using the feature subset 8. More importantly, the min–max hyperellipsoidal clusters were able to detect from the test set those anomalies that were not present in the training set.

We conducted experiments to evaluate the effect of feature subsets on the detection rates and false-positive rates for KDD 99 as well as on actual network dump data. Feature subsets 7, 8, and 12 gave excellent detection rates four each of the four categories, namely PROBE, DOS, R2L, and U2R. However, the false-positive rate for the three sets is quite high. Set 14 yields a 99.65% detection rate for normal. Therefore, using an instance of ESOM trained using set 14 to filter out the normal records, followed by an ESOM trained with feature set 8, will yield both a low false-positive rate below 0.5% at above 90% detection rate for all four categories in the KDD-99 context.

We also applied the hyperellipsoidal clustering technique to detect events in network dump data, where the header data were processed by tcptrace to extract basic and derived features of network connections. Using subsets of features from that, we were able to get detection rates between 94.05% and 99.26% at false-positive rates between 1.68% and 5.68%. Two feature subsets gave detection rates between 80% and 100% for unknown anomalous events. In this case, a detection strategy of applying an ESOM trained on the transfer feature subset followed by another ESOM trained using either the general or GH subset gives excellent results.

ACKNOWLEDGMENT

The authors would like to thank J. Huff, R. Kimbrell, and Northrop Grumman Mission Systems for facilitating this research.

REFERENCES

[1] H. I. Avi-Itzhak, J. A. Van Mieghem, and L. Rubin, "Multiple subclass pattern recognition: A maximum correlation approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 4, pp. 418–431, Apr. 1995.

[2] A. 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 Survivability Conf. and Expo.*, Hilton Head, SC, 2000, vol. II, pp. 130–144.

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

[4] A. Tikhonov and V. Arsenim, *Solutions for Ill-Posed Problems*. Washington, DC: W. H. Winston & Sons, 1977.

[5] A. Kirsch, *An Introduction to the Mathematical Theory of Inverse Problems*. New York: Springer-Verlag, 1996.

[6] M. Kudo *et al.*, "Construction of class regions by a randomized algorithm: A randomized subclass method," *Pattern Recognit.*, vol. 29, no. 4, pp. 581–588, Apr. 1996.

[7] Y. Nakamori and M. Ryoke, "Identification of fuzzy prediction models through hyper ellipsoidal clustering," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 8, pp. 1153–1173, Aug. 1998.

[8] B. Rhodes, J. Mahaffey, and J. Cannady. (2000, Oct.). "Multiple self-organizing maps for intrusion detection," *Proc. 23rd Nat. Information Systems Security Conf.*, Baltimore, MD. [Online]. Available: http://dbvis.fmi.uni-konstanz.de/members/panse/seminar_ws0203/pdf/045.pdf

[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)*, Portland, OR, 2003, pp. 1808–1813.

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

[12] 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.*, Hilton Head, SC, 2000, vol. 2, pp. 12–26.

[13] S. Sarasamma, Q. Zhu, and J. Huff, "Hierarchical Kohonen net for anomaly detection in network security," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 2, pp. 302–312, Apr. 2005.

[14] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. (2002). "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data," in *Data Mining for Security Applications*, Boston, MA. [Online]. Available: http://www1.cs.columbia.edu/ids/

[15] D. Barbara, N. Wu, and S. Jajodia. (2001). "Detecting novel network intrusions using Bayes estimators," in *Proc. 1st SIAM Int. Conf. Data Mining (SDM)*, Chicago, IL. [Online]. Available: http://ifsc.ualr.edu/wu/Paper/paper.html

[16] R. A. Maxion and K. M. C. Tan, "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. Sel. Areas Commun.*, vol. 21, no. 1, pp. 96–110, Jan. 2003.

[18] T. Lane and C. E. Brodley, "Sequence matching and learning in anomaly detection for computer security," in *Proc. AAAI Workshop: AI Approaches Fraud Detection and Risk Management*, Providence, RI, 1997, pp. 43–49.

[19] J. Oldmeadow, S. Ravinutala, and C. Leckie, "Adaptive clustering for network intrusion detection," in *Proc. 3rd Int. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD)*, Sydney, Australia, 2004, pp. 255–259.

[20] K. Leung and C. Leckie. (2005, Jan.). "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proc. 28th Australian Computer Science Conf.*, Newcastle, Australia, pp. 333–342. [Online]. Available: http://crpit.com/confpapers/CRPITV38Leung.pdf

[21] S. Ostermann. *tcptrace*. [Online]. Available: http://www.tcptrace.org/new.html

[22] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. (2003, May). "A comparative study of anomaly detection schemes in network intrusion detection," in *Proc. 3rd SIAM Conf. Data Mining*, San Francisco, CA. [Online]. Available: http://www.cs.umn.edu/research/minds/MINDS_papers.htm

[23] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-based anomaly detection: A new approach for detecting network intrusions," in *Proc. ACM Conf. Computer and Communications Security Session: Intrusion Detection*, Washington, DC, 2002, pp. 265–274.

[24] T. Kohonen, *Self-Organizing Maps*, 3rd ed, vol. 30. Berlin, Germany: Springer-Verlag, 2001.

[25] R. Nath, W. Jackson, and T. Jones, "A comparison of the classical and the linear programming approaches to the classification problem in discriminant analysis," *J. Stat. Comput. Simul.*, vol. 41, no. 1, pp. 73–93, 1992.

[26] M. Sabhnani and G. Serpen, "Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context," in *Proc. Int. Conf. Machine Learning Models, Technologies and Applications*, Las Vegas, NV, Jun. 2003, pp. 209–215.

[27] P. A. Porras and P. G. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proc. Nat. Information Systems Security Conf.*, Baltimore, MD, Oct. 1997, pp. 353–365.

[28] P. G. Neumann and P. A. Porras, "Experience with EMERALD to date," in *Proc. 1st USENIX Workshop Intrusion Detection and Network Monitoring*, Santa Clara, CA, Apr. 1999, pp. 73–80.

[29] U. Lindqvist and P. A. Porras, "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)," in *Proc. IEEE Symp. Security and Privacy*, Oakland, CA, May 1999, pp. 146–161.

[30] Q. Zhu, Y. Cai, and L. Liu, "A multiple hyper-ellipsoidal subclass model for an evolutionary classifier," *Pattern Recognit.*, vol. 34, no. 3, pp. 547–560, Mar. 2001.

[31] V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, Oct. 2004.

[32] M. Amini and R. Jalili, "Network-based intrusion detection using unsupervised adaptive resonance theory (ART)," in *Proc. 4th Int. ICSC Symp. Engineering Intelligent Systems (EIS)*, Island of Madeira, Portugal, 2004. [Online]. Available: http://www.x-cd.com/eis04/search.html

[33] S. Marsland, "Novelty detection in learning systems," *Neural Comput. Surv.*, vol. 3, pp. 157–195, 2003.

[34] A. Baker, J. Beale, B. Caswell, and M. Poore, *Snort 2.1 Intrusion Detection*, 2nd ed. Rockland, MA: Syngress, 2004.

[35] G. A. Barreto, J. C. M. Mota, L. G. M. Souza, R. A. Frota, and L. Aguayo, "Condition monitoring of 3G cellular networks through competitive neural models," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1064–1075, Sep. 2005.

[36] Z. Chen and S. Haykin, "On different facets of regularization theory," *Neural Comput.*, vol. 14, no. 12, pp. 2791–2846, Dec. 2002.

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 scientific algorithms for practical applications. Some specific areas are network intrusion detection, data mining, and collective knowledge inference techniques.

Dr. Sarasamma has been a member of the Association for Computing Machinery since 1993.

Qiuming A. Zhu (SM'97) 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. He did his postdoctoral Research in the Center for Computer Aids for Industrial Productivity at Rutgers University, and was an Assistant Professor of computer science and engineering at Oakland University from 1986 to 1990. His research interests include digital image processing and computer vision, pattern recognition, neural networks, multiagent software systems, and artificial-intelligence applications in science and engineering.