The Computer Journal Advance Access published May 12, 2013

doi:10.1093/comjnl/bxt044

MLH-IDS: A Multi-Level Hybrid Intrusion Detection Method

PRASANTA GOGOI[1], D.K. BHATTACHARYYA[1,*], B. BORAH[1] AND JUGAL K. KALITA[2]

[1]*Department of Computer Science and Engineering, Tezpur University, Napaam, Tezpur 784028, India*
[2]*Department of Computer Science, College of Engineering and Applied Science, University of Colorado, Colorado Springs, USA*
[*]*Corresponding author: dkb@tezu.ernet.in*

With the growth of networked computers and associated applications, intrusion detection has become essential to keeping networks secure. A number of intrusion detection methods have been developed for protecting computers and networks using conventional statistical methods as well as data mining methods. Data mining methods for misuse and anomaly-based intrusion detection, usually encompass supervised, unsupervised and outlier methods. It is necessary that the capabilities of intrusion detection methods be updated with the creation of new attacks. This paper proposes a multi-level hybrid intrusion detection method that uses a combination of supervised, unsupervised and outlier-based methods for improving the efficiency of detection of new and old attacks. The method is evaluated with a captured real-time flow and packet dataset called the Tezpur University intrusion detection system (TUIDS) dataset, a distributed denial of service dataset, and the benchmark intrusion dataset called the knowledge discovery and data mining Cup 1999 dataset and the new version of KDD (NSL-KDD) dataset. Experimental results are compared with existing multi-level intrusion detection methods and other classifiers. The performance of our method is very good.

Keywords: IDS; supervised; unsupervised; multi-level; outlier; features

Received 30 May 2012; revised 10 April 2013
Handling editor: George Loukas

1. INTRODUCTION

With the enormous growth of network-based computer services and the huge increase in the number of applications running on networked systems, the adoption of appropriate security measures to protect against computer and network intrusions is a crucial issue in a computing environment. Intrusions into or attacks on a computer or network system are activities or attempts to destabilize it by compromising security in confidentiality, availability or integrity of the system. As defined in [1], an intrusion detection system (IDS) monitors events occurring in a computer system or a network and analyzes them for signs of intrusions. A network-based IDS (NIDS) often consists of a set of single-purpose sensors or host computers placed at various points in a network. These units monitor network traffic, performing local analysis of that traffic and reporting attacks to a central management console.

Network-based intrusion detection is generally implemented using two approaches [2]: rule-based and anomaly-based. Rule-based (also called misuse-based) detection searches for specific patterns (or intrusion signatures given in terms of rules) in the data to effectively detect previously known intrusions. Snort [3] is a widely used rule-based NIDS that can detect intrusions based on previously known intrusion signature patterns.

The rule-based approach usually does not generate a large number of false alarms since it is based on rules that identify known intrusions but it fails to detect new types of intrusions as their signatures are not known. Anomaly detection consists of analyzing and reporting unusual behavioral patterns in computing systems. The anomaly-based detection approach typically builds a model of normal system behavior from the observed data and distinguishes any significant deviations or exceptions from this model. Anomaly-based detection implicitly assumes that any deviation from normal behavior is anomalous. The anomaly detection approach has the ability to examine new or unknown intrusions.

Based on the machine learning method used, anomaly detection can be classified into two different categories [4]: supervised and unsupervised. In supervised anomaly detection,

the normal behavior model of systems or networks is established by training with a labeled or purely normal dataset. These normal behavior models are used to classify new network connections. The system generates an alert if a connection is classified to be malign or abnormal behavior. In practice, to train a supervised anomaly-based approach, labeled or purely normal data are not easily available. It is time-consuming to acquire and error-prone to manually classify and label a large number of data instances as benign or malign.

Unsupervised anomaly detection approaches work without any training data or these models may be trained on unlabeled or unclassified data and they attempt to find intrusions lurking inside the data. The most prevalent advantage of the anomaly detection approach is the detection of unknown intrusions without any previous knowledge. However, the false detection rate tends to be higher if the behavior of some of the intrusions is not significantly different from the considered normal behavior model.

Two types of unsupervised approaches have been used to detect network anomalies: clustering and outlier-based.

Clustering is a method of grouping objects based on the similarity among them. The similarity within a cluster is high and the dissimilarity among distinct clusters is high as well. Clustering is a unsupervised method of analysis carried out on unlabeled data [5]. It results in inclusion of similar data in the same class and dissimilar data in different classes. Unsupervised anomaly-based detection usually clusters [6] the test dataset into groups of similar instances, some of which may be intrusion and others normal data. The stability of these clusters and how to label them are difficult issues. To label clusters, an unsupervised anomaly-based detection approach models normal behavior by using two assumptions [4]: (i) the number of normal instances vastly outnumbers the number of anomalies and (ii) anomalies themselves are qualitatively different from normal instances. If these assumptions hold, intrusions can be detected based on cluster size. Larger clusters correspond to normal data, and smaller clusters correspond to intrusions. But this method is likely to produce a high false detection rate as the assumptions are not always true in practice. For example, in the denial of service (DoS) category of intrusions, a large number of very similar instances are generated resulting in larger clusters than clusters corresponding to normal behavior. On the other hand, in remote to local (R2L) and user to root (U2R) categories of intrusions, legitimate and illegitimate users are difficult to distinguish and their numbers of occurrences may not be significantly different. These intrusions (U2R and R2L) may be included in normal behavior model. Consequently, these can raise the false detection rate.

The second unsupervised approach that is popular in network intrusion detection is outlier-based. *Outliers* refer to data points that are very different from the rest of the data based on appropriate measures. Such data points often contain useful information regarding unusual behavior of a system described by the data. These anomalous data points are usually called outliers. Outlier detection is widely used in the finding of anomalous activity in telecommunication, credit card fraud detection, detection of symptoms of new diseases and novel network attack detection. The major challenge for outlier detection in intrusion detection is to handle the huge volume of mixed-type, i.e. numerical and categorical, data. So, outlier detection schemes need to be computationally efficient in handling these large-sized inputs. An outlier can be an observation that is distinctly different or is at a position of abnormal distance from other values in the dataset. Detection of abnormal behavior can be based on relevant features extracted from network traces, packet or flow data. An intrusion can be detected by finding an outlier whose features are distinctly different from the rest of the data. Outliers can often be individuals or groups of objects exhibiting behavior outside the range of what is considered *normal*. A review of anomaly identification using outlier detection is available in [7].

In network-based intrusion detection, the threat may arise from new or previously unknown intrusions. The preferred detection approach for novel intrusions is anomaly-based instead of rule-based. In supervised anomaly-based detection, obtaining labeled or purely normal data is a critical issue. Unsupervised anomaly-based detection can address this issue of novel intrusion detection without prior knowledge of intrusions or purely normal data. In practice, it may be better to use a combination of supervised and unsupervised learning to achieve the best performance that current techniques allow.

1.1. Motivation

The motivation behind the design of the IDS we discuss in this paper is to develop an IDS that has a high effective overall performance for known as well as unknown attacks. The detection accuracy of a classifier is usually not equally good for each class (see Table 2 a little later in the paper). Experimental studies can be used to determine the effectiveness of a classifier in terms of its performance with known attacks so as to build an effective combination classifier to attain good overall performance with all classes of known as well as unknown attacks.

1.2. Contribution

Some of the salient contributions of the work described in this paper are given below.

(i) An enhanced supervised classifier based on a categorical clustering algorithm [8].
(ii) An unsupervised classifier based on the k-point clustering algorithm [9] with cluster stability analysis and labeling.
(iii) A supervised outlier detection algorithm based on symmetric neighborhood relationship.

TABLE 1. Categories of attack.

Category	Description
DoS	In *DoS*, an attacker tries to prevent legitimate users from using a service, viz., SYN flood, neptune and teardrop
DDoS	In distributed DoS (*DDoS*), an attacker tries to prevent legitimate users from using a service by sending packets to a network amplifier or a system supporting broadcast addressing, with the return address spoofed to the victim's IP address, viz., smurf and fraggle
R2L	In *R2L*, attackers try to gain access to a victim machine without having an account on it, viz., password guessing
U2R	In *U2R*, an attacker has local access to the victim machine and tries to gain super user privilege, viz., buffer overflow
Probe	In *Probe*, an attacker tries to gain information about the target host, viz., port-scan and ping-sweep

(iv) A hybrid multi-level classifier based on supervised, unsupervised and outlier detection algorithms, with superior performance over all classes.

1.3. Organization of the paper

The rest of the paper is organized as follows. Section 2 discusses related work in supervised, unsupervised, outlier-based and multi-level hybrid intrusion detection methods. In Section 3, we provide an overview of our approach, including a clear formation of the problem (Section 3.1), and the overall architecture of our approach (Section 3.2). Details of the approach are given in Sections 4–6. Section 4 discusses the supervised classifier we use, Section 5 the unsupervised classifier and Section 6 our outlier-based approach. These three approaches work together to achieve the best results possible. Theoretical analysis of the multi-level hybrid system is given in Section 7. Experimental results for knowledge discovery and data mining (KDD) 1999 datasets, NSL-KDD datasets and Tezpur University intrusion detection system (TUIDS) intrusion datasets are reported in Section 8. Finally, Section 9 concludes the paper.

2. RELATED WORK

Network attacks include four main categories: DoS (single source as well as distributed source), Probe, U2R and R2L, following the standard classification of network intrusions as given in [10]. The categories of attacks is given in Table 1.

Several well-cited supervised anomaly detection methods [11–18] exist in the literature. Audit data analysis and mining (ADAM) [11] is a supervised anomaly-based as well as misuse-based NIDS. ADAM uses a classifier which has been previously trained to classify the suspicious connections as a known type of attack, an unknown type or a false alarm. The method is composed of three modules: a preprocessing engine, a mining engine and a classification engine. The preprocessing engine sniffs out transport control protocol/internet protocol traffic data, and extracts information from the header of each connection according to a predefined schema. The mining engine applies mining association rules to the connection records. It works on two modes: training mode and detecting mode.

Burbeck and Nadjm-Tehrani [12] propose a clustering-based anomaly detection approach anomaly detection With fast incremental clustering (ADWICE) using an extended BIRCH [19] clustering algorithm to implement a fast, scalable and adaptive anomaly detection scheme. They apply clustering as a technique for training of the normality model.

A number of IDSes employ the unsupervised anomaly-based approach [20, 21]. Kingsly et al. [20] present an unsupervised anomaly detection method for network intrusion detection. The method uses density and grid-based [22] approach, based on a subspace clustering algorithm pMAFIA [23]. Grid-based methods divide the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on the grid structure.

Casas *et al.* [21] introduce an unsupervised network anomaly detection algorithm (UNADA) for knowledge-independent detection of anomalous traffic. UNADA uses a novel clustering method based on sub-space-density clustering to identify clusters and outliers in multiple low-dimensional spaces. The evidence of traffic structure provided by these multiple clusterings is then combined to produce an abnormality ranking of traffic flows, using a correlation-distance-based approach.

Several multi-level hybrid IDSs (MLH-IDSs) have been proposed recently to deal with the complexity of the intrusion detection problem by combining machine learning techniques. Pan *et al.* [24] combine neural networks and the C4.5 decision trees algorithm to detect a variety of attacks. Depren *et al.* [25] use a hybrid IDS consisting of an anomaly detection module, a misuse detection module and a decision support system. Zhang and Zulkernine [26] combine misuse detection and anomaly detection components using the random forests algorithm. Hwang *et al.* [27] propose a hybrid system combining signature-based intrusion detection and anomaly detection to detect novel unknown attacks. Aydyn *et al.* [28] introduce a hybrid IDS combining anomaly-based IDS and misuse-based IDS. To further increase the intrusion detection rate, as well as to simplify the algorithm, a multi-level tree classifier is proposed in [29].

In Table 2, we compare several existing IDSes based on parameters such as detection type (host-based, network-based or both), detection approach (misuse, anomaly or both), nature of detection (online or offline), nature of processing (centralized or distributed), data gathering mechanism (centralized or distributed), dataset handled and approach of analysis.

TABLE 2. List of existing IDSs.

Method	Name of IDS	Year of publication	Types of detection (H/N/Hy*)	Class (M/A/B*)	Nature of detection (R/N*)	Nature of processing (C/D*)	Data gathering mechanism (C/D*)	Datasets/ attacks handled	Approach
Supervised	Snort [3]	1999	N	M	R	C	C	KDD	Rule-based
	FIRE [58]	2000	N	A	N	C	C	KDD	Fuzzy logic
	ADAM [11]	2001	N	A	R	C	C	DARPA	Association rule
	NSOM [59]	2002	N	A	R	C	C	KDD	Neural nets
	MINDS [60]	2003	N	A	R	C	C	KDD	Classification
	FSAS [61]	2006	N	A	R	C	C	KDD	Statistical
	SVM-based [57]	2011	N	A	N	C	C	KDD	SVM & hierarchical clustering
Unsupervised	NFIDS [62]	2003	N	A	N	C	C	KDD	Neuro fuzzy logic
	HIDE [63]	2001	N	A	R	C	D	KDD	Statistical & Neural nets
	ADMIT [64]	2002	H	A	R	C	C	csh	Clustering
	N@G [65]	2003	Hy	B	R	C	C	KDD	Statistical
	ADWICE [12]	2005	N	A	R	C	C	KDD	Clustering
	DNIDS [66]	2007	N	A	R	C	C	KDD	K-NN
Outlier-based	LOF [67]	2000	N	A	N	C	C	NHL	Density-based
	Fuzzy approach [68]	2003	N	A	N	C	C	KDD	Kernel function
	RELOADED [69]	2005	N	A	N	C	C	KDD	Distance-based
	LDBSCAN [70]	2008	N	A	N	C	C	KDD	Clustering
Multi-level	Xiang *et al.* approach [29]	2004	N	B	N	C	C	KDD	Decision tree
	Depren *et al.* approach [25]	2005	N	B	N	C	C	KDD	Decision tree & neural nets
	Zhang *et al.* approach [26]	2006	N	B	N	C	C	KDD	Random forest
	Hwang *et al.* approach [27]	2007	N	B	N	C	C	KDD	Signature-based
	Hui *Lu et al.* approach [55]	2009	N	A	N	C	C	KDD	Decision tree

*H, host; N, network; Hy, hybrid; M, misuse; A, anomaly; B, both; R, realtime; N, non-realtime; C, centralised; D, distributed; KDD-KDD Cup 1999 dataset; DARPA-DARPA 1998/1999 dataset; csh-csh history file mechanism [71]; NHL-NHL96 dataset [72].

3. OVERVIEW OF OUR APPROACH

In this section, we formulate the problem we set out to solve in this paper. We follow it by a description of the general framework of our IDS.

3.1. Problem formulation

The performance of an individual classifier is not equally good for classification of all categories of attack as well as normal instances. There is a possibility of obtaining good classification accuracy for all categories in a dataset by using an appropriate combination of multiple well-performing classifiers. The problem can be formulated as given below.

For a given set of classifiers $C_1, C_2, \ldots, C_N$ and for a dataset D with p classes of instances, the objective of an MLH-IDS is to provide the best classification performance based on the effective use of those classifiers C_i that have been identified by experiments to give the best performance for a subset of classes, with reference to a set of training instances.

The objective of such a combination is to provide the best performance from the participating classifiers for each of the classes.

3.2. Proposed framework

In this paper, we present an MLH-IDS that is capable of detecting network attacks with a high degree of accuracy. The implementation of MLH-IDS uses three levels of attack detection: a supervised method, an unsupervised method and an outlier-based method.

The selection of supervised or unsupervised or outlier-based classifier at a particular level for a given dataset is based on the classification accuracy of the individual classifier for a given dataset. The overall structure of the multi-level hybrid classifier we develop is shown in Fig. 1. The first level of classification categorizes the test data into three categories DoS, Probe and Rest (unclassified). U2R and R2L, and the Normal connections are classified as Rest at this stage. This supervised algorithm is discussed in detail in Section 4. The main purpose at level 1 is to extract as many U2R, R2L and Normal connections as possible accurately from the data using a supervised classifier model. This is because U2R and R2L are generally quite similar to normal connections. DoS and Probe attack connections on the other hand are generally more different. The second level splits Rest into Normal and Rest categories. At level 2, the Rest category is classified as Normal and Rest (i.e. class containing attacks) using an unsupervised classifier model. The unsupervised classifier used in the second level is discussed in Section 5. The third level separates Rest into U2R and R2L. The U2R category data are extracted from Rest using the outlier-based classifier model and the remaining elements in the Rest category are classified as R2L. The outlier-based classifier is discussed in Section 6. In high-dimensional data, feature selection using mutual information (MI) is a widely accepted approach. In level 3, MI-based relevant features [30] of U2R are used in the model to classify the U2R category attacks. Other than U2R category records, the remaining items are classified as R2L category attacks.

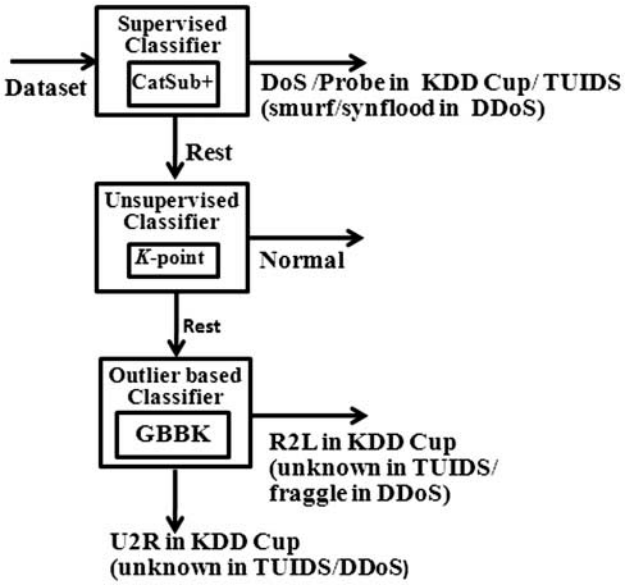

FIGURE 1. Architecture of a multi-level hybrid classifier.

4. SUPERVISED CLASSIFICATION

Our classification technique creates a set of representative clusters from the available labeled training objects. Unlabeled test objects are then inserted in these representative clusters based on similarity calculation and thus they get labels of the clusters in which they are inserted. The clustering algorithm we use is a modification of the algorithm presented by the authors in [8]. The notations used in the algorithm are given in Table 3.

Let the dataset to be clustered contain n objects, each described by d attributes $A_1, A_2, \ldots, A_d$ having finite discrete-valued domains $D_1, D_2, \ldots, D_d$, respectively. A data object is represented as $X = \{x_1, x_2, \ldots, x_d\}$. The jth component of object X is x_j and it takes one of the possible values defined in domain D_j of attribute A_j. Referring to each object by its serial number, the dataset can be represented by the set $N = \{1, 2, \ldots, n\}$. Similarly, the attributes are represented by the set $M = \{1, 2, \ldots, d\}$.

The *similarity between two data objects X and Y* is the sum of per attribute similarity for all the attributes. It is computed as

$$sim(X, Y) = \sum_{j=1}^{d} s(x_j, y_j), \quad (1)$$

TABLE 3. Algorithm notation.

Symbol	Description
M	Attribute size of the dataset
k	Number of randomly selected records
S	Set of clusters
MinAtt, *MinmAtt*	Threshold value for minimum attribute
minSize	Threshold value for minimum objects in a cluster
$sim(X, Y)$, *simO*	Similarity value between two objects
$sim(C, X)$, *simC*	Similarity value between a cluster and an object
δ_j, δ_{aj}	Similarity threshold
Olist, *Oblist*	List of objects forming a cluster
Profile, *Clusprofile*	Profile of a cluster
noAttribtes, N_{attrib}	The cardinality of the subset of the attributes contributing in forming a cluster
Dattributes, D_{attrib}	Distribution of position of attributes in a cluster profile
Values, V_{attrib}	Values of attributes forming a cluster profile

where $s(x_j, y_j)$ is the similarity for the jth attribute defined as

$$s(x_j, y_j) = \begin{cases} 1 & \text{if } |x_j - y_j| \leq \delta_j, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where δ_j is the similarity threshold for the jth attribute. For categorical attributes $\delta_j = 0$ and for numeric attributes $\delta_j \geq 0$.

We use a subspace-based incremental clustering technique. A cluster is a set of objects that are similar over a subset of attributes only. The minimum size of the subset of attributes required to form a cluster is defined by the threshold *MinAtt*. Let the subset of defining attributes be represented by $Dattributes = \{a_1, a_2, \ldots, a_{noAttributes}\}$ such that $Dattributes \subseteq M$ and *noAttributes* is the size of *Dattributes*. A cluster is represented by its profile. Our profile representation is similar to that of an object. All objects in a cluster are similar with respect to the profile. The cluster profile is defined by a set of values, $Values = \{v_1, v_2, \ldots, v_{noAttributes}\}$ taken over the corresponding attributes in *Dattributes*, that is $v_1 \in D_{a_1}$ is the value for attribute $a_1 \in M$, $v_2 \in D_{a_2}$ is the value for attribute $a_2 \in M$ and so on. Thus, the cluster profile is defined by

$$Profile = \{noAttributes, Dattributes, Values\}. \tag{3}$$

Let $Olist \subseteq N$ be the list of data objects in the cluster. A cluster C is completely defined by its *Profile* and *Olist*:

$$C = \{Olist, Profile\}. \tag{4}$$

The subspace-based incremental clustering algorithm inserts an object in any one of the set of clusters existing at a particular moment. So the similarity between a cluster and a data object needs to be computed. Obviously, the cluster profile is used for computing this similarity. As the similarity needs to be computed over the set of attributes in *Dattributes* only, the *similarity function between a cluster C and an object X* becomes

$$sim(C, X) = \sum_{j=1}^{noAttributes} s(v_j, x_{a_j}), \tag{5}$$

where

$$s(v_j, x_{a_j}) = \begin{cases} 1 & \text{if } |v_j - x_{a_j}| \leq \delta_{a_j}, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

To obtain appropriate values for δ_j and δ_{a_j}, the following two tasks are performed.

(i) Preprocessing of the dataset using logarithm to the base 2 normalization [31] to remove bias. We discretize continuous-valued attributes by taking logarithm to the base 2 and then converting to integer. This is done for each attribute value z using the computation: $if\,(z > 2) z = \int(\log_2(z)) + 1$. Before taking the logarithm, the attributes that take fractional values in the range [0, 1] are multiplied by 100 so that they take values in the range [0, 100].
(ii) Use of a heuristic method to identify appropriate range for δ_j and δ_{a_j} by exhaustive experiments using benchmark and real-life datasets.

A discussion on the selection of the appropriate threshold value for δ_j in Equation (2) and δ_{a_j} in Equation (6) is given in Section 8.3.

TABLE 4. A sample dataset.

S. No.	A_1	A_2	A_3	A_4	A_5
1	a_3	b_2	c_4	d_1	e_2
2	a_2	b_2	c_4	d_3	e_2
3	a_3	b_1	c_2	d_1	e_1
4	a_2	b_2	c_4	d_1	e_2
5	a_3	b_1	c_2	d_3	e_1
6	a_1	b_2	c_1	d_2	e_2
7	a_3	b_1	c_2	d_2	e_1

EXAMPLE 1. Consider a small dataset shown in Table 4 with seven objects defined over five attributes A_1, A_2, A_3, A_4 and A_5. The domains for the attributes are, respectively, $D_1 = \{a_1, a_2, a_3\}$, $D_2 = \{b_1, b_2\}$, $D_3 = \{c_1, c_2, c_3, c_4\}$, $D_4 = \{d_1, d_2, d_3\}$ and $D_5 = \{e_1, e_2\}$.

Clusters C_1 and C_2 can be identified in the dataset with $MinAtt = 3$ and $\delta_{a_j} = 0$ for $a_j \in Dattributes$:

$$C_1 = \{Olist = \{1, 2, 4\},\ noAttributes = 3,\ Dattributes = \{2, 3, 5\},\ Values = \{b_2, c_4, e_2\}\},$$
$$C_2 = \{Olist = \{3, 5, 7\},\ noAttributes = 4,\ Dattributes = \{1, 2, 3, 5\},\ Values = \{a_3, b_1, c_2, e_1\}\}.$$

4.1. The *CatSub+* algorithm

Our algorithm, which is an enhanced version of the *CatSub* algorithm [8], is named *CatSub+*. Unlike the *CatSub* algorithm, we minimize the dependency on input parameters by providing the probable range of parameter values based on a heuristic method. Further, *CatSub+* is cost effective, since it works on subset of relevant features selected using an information-theoretic method [32]. *CatSub+* starts with an initially empty set of clusters. It reads each object X_i sequentially, inserts it in an existing cluster based upon the similarity between X_i and the clusters, or a new cluster is created with X_i if it is not similar enough, as defined by the threshold *MinAtt*, to be inserted in any one of the existing clusters. Search for a cluster for inserting the present object is started at the last cluster created and moves toward the first cluster until the search is successful. If successful, the object is inserted in the cluster found and the search is terminated. At the time of inserting the object in the found cluster C, the values of the defining attributes of the cluster (*C.noAttributes*) are set according to the computed similarity measure between the cluster and the object. The sets of *C.Dattributes* along with *C.Values* are also updated. If the search is not successful, a new cluster is created and the object itself made the representative object of the cluster, i.e. the full set of attributes becomes *Dattributes* while the full set of values of the object becomes corresponding *Values* of the new cluster profile.

Initially, *CatSub+* is trained with a fixed number of known clusters. Once the clusters and corresponding profiles are built for the known classes, newer instances are incrementally inserted in any one of the clusters.

Before the initiation of cluster formation and respective profile building, all the unselected objects are marked as unprocessed. Similarity thresholds *minAtt* and *minSize* are assigned high values and they are gradually decreased in steps. In each iteration, the remaining unprocessed objects are clustered using the similarity measure, with reference to δ. If it fails to insert an object in any of the preexisting known clusters (created in the previous iteration), then a new cluster is created with the object. When the clustering process ends in the present iteration, cluster profiles are extracted from each of the clusters having at least *minSize* objects in it and the objects in such a cluster are marked as processed. All insignificant clusters, whose sizes are less than *minSize*, are deleted. The remaining new clusters become known clusters for the next iteration after making them empty by deleting their object lists. Then the threshold values *minSize* and *minAtt* are reduced so that the next iteration can create larger clusters instead of fragmented clusters. By reducing the thresholds, more generalization is allowed. The algorithm iterates so long as there are unprocessed objects remaining. To ensure termination of the algorithm, *minSize* is reduced to *minSize*/2 so that the ultimate value of *minSize* becomes 1, after which no objects will remain unprocessed. The threshold *minAtt* is loosened by setting $minAtt = minAtt - \alpha$, where α is a small integral constant such as 1 or 2. Reduction of *minAtt* below a certain level (*MIN*) is not allowed. It remains constant at *MIN*. Generalization beyond the MIN level will make data objects belonging to two different classes indistinguishable. When the clustering process terminates, the set of profiles found in the profile file becomes the final cluster profiles for use in the prediction process.

CatSub+ differs from *CatSub* in the following ways:

(i) It minimizes the dependency on input parameters by providing the probable range of δ_j and δ_{aj} based on a heuristic method.
(ii) It works on a subset of relevant features selected based on an information-theoretic method [32].

The training algorithm is given as Algorithm 1. The algorithm for prediction is given as Algorithm 2.

Algorithm 1 Similarity computation and update functions of *CatSub+*

```
1:  Function sim (cluster C, object X)
2:  count = 0;
3:  k = C.noAttributes;
4:  for (j = 0; j < k; j = j + 1) do
5:     l = C.a[j];
6:     if (abs(C.v[j] − x[l]) <= δ[l]) then
7:        count = count + 1;
8:     end if
9:  end for
10: return count;
11:
12: Function update (cluster C, int r, object X)
13: append(r, C.Olist);
14: count = 1;
15: m = C.noAttributes;
16: for (j = 0; j < m; + + j) do
17:    l = C.a[j];
18:    if (abs(s(C.v[j] − x[l]) <= δ[l]) then
19:       count = count + 1;
20:       C.v[count] = C.v[j];
21:       C.a[count] = C.a[j];
22:    end if
23:    C.noAttributes = count;
24: end for
```

Algorithm 2 Prediction algorithm of *CatSub+*

```
1: file1=open("TrainProfile","read");
2: file2=open("DataFile","read");
3: read(file1, totTrain);
4: for (i = 0; i < totTrain; i = i + 1) do
5:    read(file1, k);
6:    C[i].noAttributes = k;
7:    for (j = 0; j < k; j = j + 1) do
8:       read(file1, C[i].a[j]);
9:       for (j = 0; j < k; j = j + 1) do
10:         read(file1, C[i].v[j]);
11:         if count == C[j].noAttributes then
12:            print("Object" i " is of category"
13:            clusterLabel[j]);
14:            found = 1; break;
15:         else
16:            if count > max then
17:               max = count; position = j;
18:            end if
19:         end if
20:      end for
21:      if found == 0 then
22:         print("Object" i "is of category";
23:         clusterLabel[position]);
24:      end if
25:   end for
26: end for
```

4.2. Complexity analysis

The clustering algorithm requires one pass through the set of training examples that currently remain unprocessed. Each training example needs to be compared with existing clusters one after another until it gets inserted in one of the clusters. The similarity computation involves a subset of attributes. Therefore, the clustering process has a complexity $O(ncd)$, where n is the number of training examples, c is the number of clusters and d is the number of attributes. Each of the created clusters needs to be visited to extract its size and profile. Hence, maximum time complexity of one iteration of the training algorithm becomes $O(ncd) + O(c)$. The algorithm performs at most k iterations, where $k = \log_2(minSize)$. As *minSize* is the minimum number of objects for a cluster to be considered significant, it is not large. The overall maximum time complexity of the algorithm is $O(kncd) + O(kc)$.

5. UNSUPERVISED CLASSIFICATION

The unsupervised classification method uses the k-point algorithm [9] to create a set of representative clusters from the available unlabeled objects in the data. Initially, the method considers k objects randomly from the dataset. The data objects are then gathered around these selected points (or objects) based on similarities considering various attributes. The clusters are formed using two similarity measures: (i) similarity between two objects and (ii) similarity between a cluster and an object. The description of notations used in the algorithm is given in Table 3.

TABLE 5. A sample dataset.

S. No.	A_1	A_2	A_3	A_4	A_5	A_6
1	a_3	b_2	c_4	d_1	e_2	f_1
2	a_2	b_2	c_4	d_3	e_2	f_2
3	a_3	b_1	c_2	d_1	e_1	f_2
4	a_2	b_2	c_4	d_1	e_2	f_1
5	a_3	b_1	c_2	d_3	e_1	f_3
6	a_1	b_2	c_1	d_2	e_2	f_1
7	a_3	b_1	c_2	d_2	e_1	f_3
8	a_2	b_2	c_4	d_3	e_2	f_2
9	a_3	b_1	c_2	d_1	e_1	f_4
10	a_3	b_1	c_2	d_1	e_1	f_5

EXAMPLE 2. Consider a sample dataset shown in Table 5 with 10 objects defined over 6 attributes A_1, A_2, A_3, A_4, A_5 and A_6. The domains for the attributes are, respectively, $D_1 = \{a_1, a_2, a_3\}$, $D_2 = \{b_1, b_2\}$, $D_3 = \{c_1, c_2, c_3, c_4\}$, $D_4 = \{d_1, d_2, d_3\}$, $D_5 = \{e_1, e_2\}$ and $D_6 = \{f_1, f_2, f_3, f_4, f_5\}$.

Clusters C_1, C_2, C_3 and C_4 can be identified in the dataset with $MinAtt = 6/2 = 3$

$C_1 = \{Oblist = \{2, 8\}, N_{\text{attrib}} = 6,$
$D_{\text{attrib}} = \{1, 2, 3, 4, 5, 6\}, V_{\text{attrib}} = \{a_2, b_2, c_4, d_3, e_2, f_1\}\},$
$C_2 = \{Oblist = \{1, 4\}, N_{\text{attrib}} = 5,$
$D_{\text{attrib}} = \{2, 3, 4, 5, 6\}, V_{\text{attrib}} = \{b_2, c_4, d_1, e_2, f_1\}\},$
$C_3 = \{Oblist = \{5, 7\}, N_{\text{attrib}} = 5,$
$D_{\text{attrib}} = \{1, 2, 3, 5, 6\}, V_{\text{attrib}} = \{a_3, b_1, c_2, e_1, f_3\}\},$
$C_4 = \{Oblist = \{3, 9, 10\}, N_{\text{attrib}} = 5,$
$D_{\text{attrib}} = \{1, 2, 3, 4, 5\}, V_{\text{attrib}} = \{a_3, b_1, c_2, d_1, e_1\}\}.$

5.1. The *k*-point algorithm

The unsupervised classification algorithm starts with an empty set of clusters. Initially, k objects are selected randomly from the dataset. It reads each object X_i sequentially from the dataset and inserts it in an existing cluster based upon the similarity between X_i and a cluster. If the similarity between a cluster and the object is below a threshold, a new cluster is created with X_i, if X_i is similar with any of the randomly selected k objects for a defined threshold *MinmAtt* of attributes. The search for a cluster for inserting an object starts from the beginning of the created cluster set until the search is successful. The objects that

are neither possible to include in any one of the clusters nor to create a new cluster based upon the specified value of *MinmAtt* are excluded from the clusters. Based upon the similarity among the profiles, similar clusters are merged into a single cluster. The largest cluster is selected and labeled as normal on the basis of the assumption of the normal behavior model. The algorithm is given as Algorithm 3.

Algorithm 3 k-point Algorithm

Input: Dataset D,Value of k;
Output: Largest Cluster, C_L;

Select k records randomly from the D;
Find number of attributes M of each of k records
$S = \emptyset$; $MinmAtt = M/2$; $T = M$;
if $D \neq \emptyset$ **then**
  Fetch a record d from D;
end if
if d is unselected record and $S \neq \emptyset$ **then**
  Find a cluster C_i from S ;
  Compute similarity, $simC(C_i, d)$ between cluster C_i and object d;
else
  Goto step 5;
end if
if $simC(C_i, d) == 0$ **then**
  Add record d to C_i;
  Go to step 5;
end if
Compute similarity, $simO(d, k_i)$ between object d and any record k_i from random records for T attributes;
if $simO(d, k_i) == 0$ **then**
  Create a cluster C_j with $Clusprofile = (N_{attrib}, D_{attrib}, V_{attrib})$;
  Include C_j to S;
  Go to step 5;
else
  $T = T - 1$;
end if
if $T > MinmAtt$ **then**
  Go to step 17;
else
  Go to step 5;
end if
if Number of clusters in $S > 2$ **then**
  if Profiles of $C_x ==$ Profile of C_y **then**
    Merge *Oblist* of C_y with *Oblist* of C_x;
    Update S;
  end if
end if
Find the largest cluster C_L by comparing *Oblist* of clusters from S;
Stop;

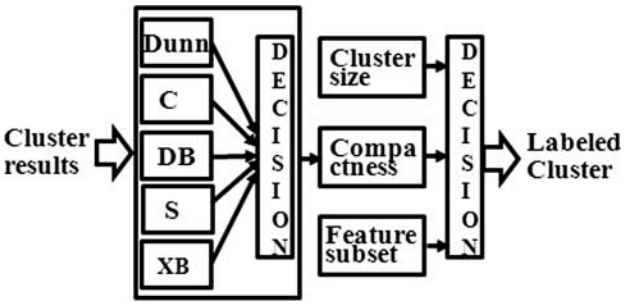

FIGURE 2. Architecture of multiple indices cluster labeling.

TABLE 6. Cluster stability criteria.

Stability measure	Range of value	Criteria for better cluster
Dunn index	(0,∞)	Maximized
C-index	(0,1)	Minimized
DB's index	(0,∞)	Minimized
Silhouette index	(−1,1)	Near 1
XB index	(0,1)	Minimized

5.2. Complexity analysis

The k-point algorithm requires one pass through the dataset. Each object needs to be compared with existing clusters one after another until it gets inserted in one of the clusters. The similarity computation involves a subset of attributes. Therefore, the clustering process has a complexity $O(ncd)$, where n is the number of objects in the dataset, c is the number of clusters and d is the number of attributes. Each of the created clusters needs to be visited for k number of objects for d attributes. Hence, the maximum time complexity of k-point algorithm becomes $O(ncd) + O(kd)$.

5.3. Cluster labeling technique

We analyze the stability of the clusters obtained using the k-point algorithm. We propose an ensemble-based stability analysis technique based on the Dunn index (Dunn) [33], C-index (C) [34], Davies Bouldin's index (DB) [35], Silhouette index (S) [36] and Xie–Beni index (XB) [37] for cluster validity (shown in Fig. 2).

The criteria used for labeling the cluster stability analysis are given in Table 6. Cluster results are passed to a function to compute all the indices for each of the clusters $C_1, C_2, \ldots, C_k$, and if the obtained value is within the acceptable range, then the cluster index, CI_i is stored as 1; otherwise it is assigned 0, as defined below.

$$CI_i = \begin{cases} 1, & j\text{th index value, } I_j \text{ is within its valid range,} \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

TABLE 7. Outlier detection algorithm notation.

Symbol	Description
$\|D\|$	Size of dataset
k	Size of neighborhood objects
p, q	Any objects
$dist(p, q)$	Distance between objects p and q
$NNk(p)$	Nearest neighbor set of k objects for object p
$getNNk()$	Function for computing nearest neighbor set of k objects
$FNNk(p)$	Forward nearest neighbor set of k objects for object p
$FNOFk(p)$	Forward neighbor outlier factor for object p
$getFNOFk()$	Function for computing forward neighbor outlier factor
M	Threshold value for forward neighbor outlier factor

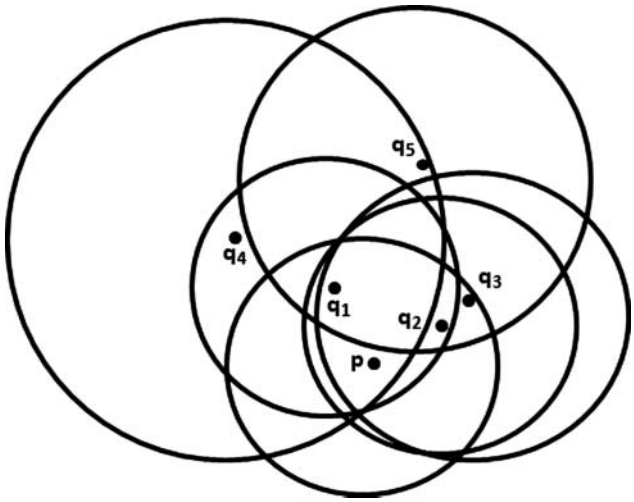

FIGURE 3. Example 3 dataset plot.

Finally, we consider the number of occurrences of 1 to decide the compactness of a cluster. This becomes input to the subsequent labeling task along with the other two measures, namely that, cluster size and dominant feature subset shown in Fig. 2. To compute the dominating feature subset, we use a rough set and a modified genetic algorithm-based method described in [38].

6. OUTLIER MINING

We have developed an outlier mining method based on symmetric neighborhood relationships [39]. For each object of the dataset, a forward neighbor outlier factor is estimated by finding the nearest neighbor set and the forward nearest neighbor set of the data objects to identify outliers. The description of notation used in the algorithm is given in Table 7.

In the dataset $D = \{d_1, d_2, \ldots, d_n\}$ of n objects, let d_i and d_j be two arbitrary objects in D. We use Euclidean distance to evaluate the distance between objects d_i and d_j, denoted as $dist(d_i, d_j)$.

DEFINITION 1. *The Nearest Neighbor Set of k objects for an object p ($NNk(p)$) is the set of k nearest neighbor objects of p where $k > 0$. In dataset D of $|D|$ objects, $|NNk(p)| = k$*

(i) *if* $\forall p \notin NNk(p)$ *and*
(ii) $dist(o, p) < dist(ó, p)$ *where o and ó are kth and* $(k+1)$*th nearest neighbors of p, respectively.*

DEFINITION 2. *Forward Nearest Neighbor Set of k objects of object p is the set of objects whose NNk contains p, denoted as $FNNk(p)$. In dataset D of $|D|$ objects where p and q are arbitrary objects, $FNNk(p)$ is defined as*

$$FNNk(p) = \{q \in D \mid p \in NNk(q), p \neq q\}. \quad (8)$$

A score for each object of the dataset is computed based on $|NNk|$ and $|FNNk|$ of the object. The score is termed Forward Neighbor Outlier Factor of k objects ($FNOFk$) and it decides the strength of linkage of the object with other objects.

DEFINITION 3. *The Forward Neighbor Outlier Factor of k objects for an object p is the ratio of the remaining number of objects of $FNNk(p)$ of the dataset D (except object p) to the number of dataset objects (except object p), denoted as $FNOFk(p)$. In dataset D of objects $|D|$, $FNOFk(p)$ is defined as*

$$FNOFk(p) = \frac{|D| - |FNNk(p)| - 1}{|D| - 1} = 1 - \frac{|FNNk(p)|}{|D| - 1}. \quad (9)$$

EXAMPLE 3. *A sample 2D dataset is plotted in Fig. 3. The dataset has six objects p, q_1, q_2, q_3, q_4 and q_5. For neighborhood size $k = 3$, the NNk and FNNk will be as follows*:

$$\begin{aligned}
NNk(p) &= \{q_1, q_2, q_3\}, & FNNk(p) &= \{q_1, q_2, q_3, q_4\},\\
NNk(q_1) &= \{p, q_2, q_4\}, & FNNk(q_1) &= \{p, q_2, q_3, q_4, q_5\},\\
NNk(q_2) &= \{p, q_1, q_3\}, & FNNk(q_2) &= \{p, q_1, q_3, q_5\},\\
NNk(q_3) &= \{p, q_1, q_2\}, & FNNk(q_3) &= \{p, q_2, q_5\},\\
NNk(q_4) &= \{p, q_1, q_5\}, & FNNk(q_4) &= \{q_1\},\\
NNk(q_5) &= \{q_1, q_2, q_3\}, & FNNk(q_5) &= \{q_4\}.
\end{aligned}$$

The number of objects of NNk and FNNk for $k = 3$ are as follows:

$$\begin{aligned} |NNk(p)| &= 3, \quad |FNNk(p)| = 4, \\ |NNk(q_1)| &= 3, \quad |FNNk(q_1)| = 5, \\ |NNk(q_2)| &= 3, \quad |FNNk(q_2)| = 4, \\ |NNk(q_3)| &= 3, \quad |FNNk(q_3)| = 3, \\ |NNk(q_4)| &= 3, \quad |FNNk(q_4)| = 1, \\ |NNk(q_5)| &= 3, \quad |FNNk(q_5)| = 1. \end{aligned}$$

The outlier factor $FNOFk$ for $k = 3$ and $|D|$=6 are as follows:

$$\begin{aligned} FNOFk(p) &= (1 - \tfrac{4}{5}) = 0.20, \\ FNOFk(q_1) &= (1 - \tfrac{5}{5}) = 0.0, \\ OFk(q_2) &= (1 - \tfrac{4}{5}) = 0.20, \\ FNOFk(q_3) &= (1 - \tfrac{3}{5}) = 0.40, \\ FNOFk(q_4) &= (1 - \tfrac{1}{5}) = 0.80, \\ FNOFk(q_5) &= (1 - \tfrac{1}{5}) = 0.80. \end{aligned}$$

The outliers are found for threshold $M \geq 0.80$ and $k = 3$ as follows:

$$FNOFk(q_4) = 0.80 \quad \text{and} \quad FNOFk(q_5) = 0.80.$$

6.1. GBBK algorithm

The outlier detection algorithm named the GBBK algorithm consists of two functions: $getFNOFk(D, k)$ and $getNNk(D, p, k)$ (reported as Algorithm 4). The function $getFNOFk(D, k)$ computes the distances among all objects using Euclidean distance, sorts all distances and searches for the shortest k distances. The function $getNNk(D, p, k)$ searches for forward nearest neighbor objects for each of k objects returned by the function $getFNOFk(D, k)$ for any object and computes a score using Equation (9).

6.2. Complexity analysis

The outlier detection algorithm comprises two functions: *getNNk*() and *getFNOF*(). The complexity of function *getNNk*() is due to the distance computation among n objects and sorting of n object distances using Quicksort. So, it is $n \log n$. Thus, the time complexity of this function is $O(n + n \log n)$. Again, the complexity of the function *getFNOF*() is due to search among $n \times k$ objects, computation of the outlier factor for n objects and search of n objects using a user-defined threshold value. Thus, the time complexity of this function is $O(n \times k \times (n + n \log n) + 2n)$. Hence, the time complexity of the outlier algorithm is $O(n \times k \times (n + n \log n) + 2n) \cong O(n^2)$. However, the use of spatial index structure [40] reduces the complexity significantly.

Algorithm 4 GBBK algorithm

Input: *Dataset D*, *M*, *k*, *p*;
Output: *Outlier List L*;
1: $X = getFNOFk(D, k)$;
2: $\forall_{t \in X}$
3: **if** $t \geq M$ **then**
4: Add t to L;
5: **end if**
6:
7: Function $getFNOFkD, k$
8: **while** $|D| \neq Null$ **do**
9: $\forall_{p \in D}\ S = getNNk(D, p, k)$;
10: **end while**
11: $\forall_{q \in S}\ T = getNNK(D, q, k)$;
12: **if** $p \in T$ **then**
13: Add q to list of $FNNk(p)$;
14: $|FNNk(p)| = |FNNk(p)|$+1;
15: **end if**
16: Compute $\forall_{p \in D} FNOFk(p) = \{1 - \frac{|FNNk(p)|}{|D|-1}\}$;
17: **return** $FNOFk(p)$;
18:
19: Function $getNNkD, p, k$
20: **if** $|D| \neq Null$ **then**
21: $\forall_{q; p \neq q; p, q \in D}$ Compute $dist(p, q)$;
22: **end if**
23: $\forall_q$ Sort $dist(p, q)$;
24: Add k shortest distant objects from p to $NNk(p)$;
25: **return** $NNk(p)$;

7. MLH-IDS: COMPLEXITY ANALYSIS

The multi-level hybrid intrusion detection method (MLH-IDS) is composed of three individual methods we have already discussed in this paper. The time complexities of each method are (i) $O(ncd) + O(c)$ for *CatSub+* (reported in Section 4.2), (ii) $O(ncd)$ for k-point (reported in Section 5.2) and (iii) $O(n^2)$ for the outlier-based method (reported in Section 6.2). The time complexity of MLH-IDS is the summarization of these three methods.

LEMMA 1. *The performance of MLH-IDS in terms of detection accuracy cannot be less than the performance of the individual classifiers used.*

Proof. Let the MLH-IDS be called S for our current discussion. Thus, S is comprised of m classifiers C_1,C_2,...,C_m, which are placed at level 1, level 2,..., level m, respectively. Also, let $p < p_i$, where p is the performance of S and p_i is the performance of classifier C_i ($i = 1, 2, \ldots, m$). As per strategy (as reported in Section 3), C_i is placed at a level k ($k = 1, 2, \ldots, m$) for detection of certain class(es) for which its performance is the best among the m classifiers and performance of S is the performance of the classifiers C_i used at that level. In

TABLE 8. TUIDS intrusion dataset *packet level*.

Data sets	DoS	Probe	Normal	Total
Packet Level	151 203	26 839	254 833	432 875
Packet Level+	75 600	13 420	127 416	216 436

TABLE 9. TUIDS intrusion dataset *flow level*.

Data sets	DoS	Probe	Normal	Total
Flow Level	162 333	62 741	175 057	400 131
Flow Level+	81 166	31 375	87 528	200 069

TABLE 10. Attack list and the generating tools.

Attack	Generation tool	Attack	Generation tool
bonk	*targa2.c*	1234	*targa2.c*
jolt	*targa2.c*	*saihyousen*	*targa2.c*
land	*targa2.c*	*oshare*	*targa2.c*
nestea	*targa2.c*	*window*	*targa2.c*
newtear	*targa2.c*	*syn*	*Nmap*
syndrop	*targa2.c*	*xmas*	*Nmap*
teardrop	*targa2.c*	*fraggle*	*fraggle.c*
winnuke	*targa2.c*	*smurf*	*smurf4.c*

other words, performance of S at level k cannot be less than that of the classifier used at level k, since the strategy is followed for each level, and so $p \not< p_i$. Thus, there is a contradiction and hence the proof. □

8. EXPERIMENTAL RESULTS

In this section, we report a comprehensive experimental evaluation of the efficiency and effectiveness of the MLH-IDS method. All necessary experiments were carried out on an Intel workstation with Intel core 2 Quad @2.4GHz, 2 GB RAM, 160GB HDD. The programs were developed in *C* in a Linux environment. The method was evaluated with our real-time TUIDS [41] intrusion dataset, a *DDoS Dataset*, the KDD Cup 1999 dataset [42] and the new version of KDD (NSL-KDD) dataset [43].

8.1. Datasets

The description of the four datasets is given next.

8.1.1. TUIDS intrusion dataset

The MLH-IDS method was evaluated using our own real-life *TUIDS* intrusion dataset. A summary of the dataset is given in Tables 8 and 9. The TUIDS dataset actually includes two datasets—*Packet Level* and *Flow Level*. The network traffic for attack and normal was captured using our local network within a 4-week period. The attacks were generated using attack tools [44] against a local network server and the produced traffic was collected as known attack traffic. Sixteen different types of attacks were generated. The attacks along with the corresponding tools for their generation are given in Table 10. The network traffic was captured at packet level and flow level through two separate port-mirroring machines. The captured data were preprocessed and filtered to extract various types of features.

The setup of the testbed for network flow capture included one router, one L3 switch, two L2 switches, one server, two workstations and forty nodes. A number of (six) virtual local area networks (VLANs) were created from the L3 switch and the L2 switch and nodes and workstations were connected to separated VLANs. The L3 switch was connected to a router through an internal IP router and the router was connected to the Internet through an external IP router. The server was connected to the L3 switch through the mirror port to observe the traffic activity to the switch. Another local area network (LAN) of 350 nodes was connected to other VLANs through five L3 and L2 switches and three routers. The attacks were launched within our testbed as well as from another LAN through the Internet. In launching attacks within the testbed, nodes of one VLAN were attacked from nodes of another VLAN and also the same VLAN. Normal traffic was created within our testbed under restricted conditions after disconnecting another LAN. The traffic activities to our testbed were observed in the computer connected to the mirror port. A diagram of the testbed for generation of real-life intrusion datasets is shown in Fig. 4.

The packet level network traffic was captured using the open source software tool called *gulp* [45]. The packets were analyzed using the open source packet analyzing software *wireshark* [46]. The raw packet data were preprocessed and filtered before extracting and constructing new features. In packet level network traffic, 50 types of features were extracted. The list of features are given in Table 11. The extracted features are classified into four groups: (i) *basic*, (ii) *content-based*, (iii) *time-based* and (iv) *connection-based*.

The network flow is a unidirectional sequence of packets passing through an observation point in the network during a certain time interval between source and destination hosts. All traffic belonging to a particular flow has a set of common properties. The NetFlow protocol (IPFIX standard) [47, 48] provides a summarization of information about the router or switch traffic. We used the NetFlow version 5 protocol [49] to export flow records and used *nfdump* [50] to receive flow records. The extracted flow-level features are reported in Table 12. The extracted features are of 24 types and are classified into three groups: (i) *basic*, (ii) *time-window-based* and (iii) *connection-based*.

8.1.2. DDoS dataset

The MLH-IDS method was evaluated with a *DDoS Dataset* we generated as well. Details of this dataset are given in

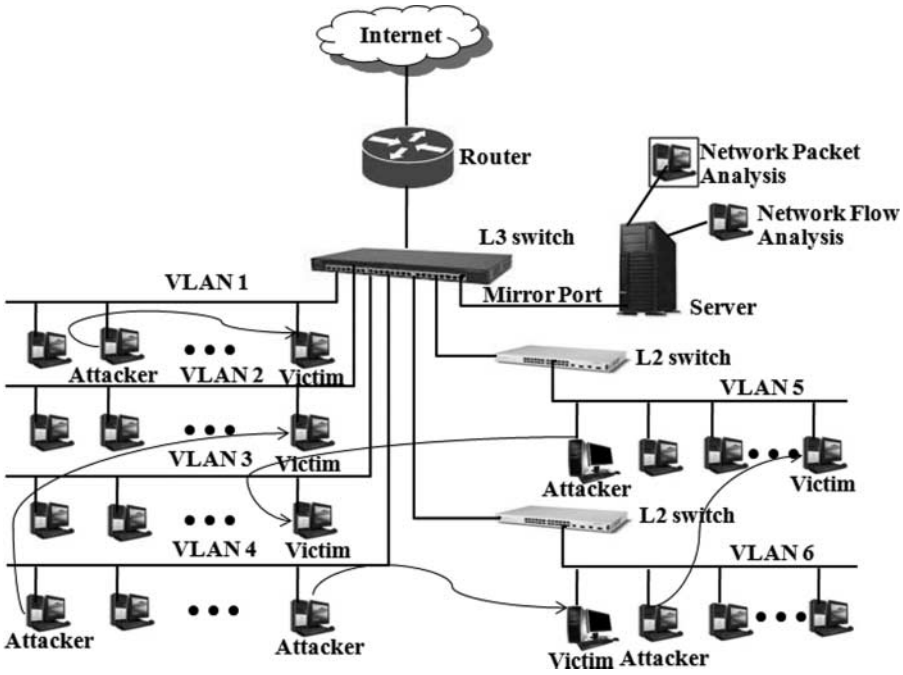

FIGURE 4. Testbed for generation of TUIDS intrusion dataset.

Table 13. The experimental testbed shown in Fig. 4 was used for generation of DDoS attacks and normal network traffic. Attacks were generated from one VLAN to another by spoofing a node of one intermediate VLAN using attack tools [44]. The network traffic was captured during a period of three weeks. We used the NetFlow version 5 protocol to export flow records and used *nfdump* to receive flow records. The extracted *DDoS Dataset* features are given in Table 14.

8.1.3. *KDD Cup 1999 dataset*

The MLH-IDS method was also tested on datasets available in the KDD Cup 1999 intrusion detection benchmark datasets [42]. Each record of the datasets represents a connection between two network hosts according to network protocols and is described by 41 attributes (38 continuous or discrete numerical attributes and 3 categorical attributes). Each record is labeled as either normal or one specific kind of attack. Each dataset contains 22 types of attacks. The attacks fall in one of the four categories: $U2R$, $R2L$, *DoS* and *Probe*. The number of samples of each category of attack in the *Corrected KDD* dataset and the 10% *KDD* dataset are shown in Table 15.

8.1.4. *NSL-KDD datasets*

NSL-KDD is a network-based intrusion dataset [43]. It is a filtered version of KDD Cup 1999 intrusion detection benchmark dataset. In the KDD Cup 1999 dataset, there are a huge number of redundant records, which can cause the learning algorithms to be biased toward the frequent records. To solve this issue, one copy of each record was kept in the NSL-KDD dataset. Statistical characteristics of the two datasets of NSL-KDD: (i) $KDDTrain^+$ and (ii) $KDDTest^+$ are shown in Table 16.

8.2. Data preprocessing

We discretized the continuous valued attributes by taking logarithm to the base 2 and then converting to integer as described in Section 4. Nominal valued attributes are mapped to discrete numerical codes which are nothing but serial numbers (beginning with zero) of unique attribute values in the order in which they appear in the dataset. The class label attribute is removed from the dataset and stored separately in a different file. The class labels are used for training and evaluating the detection performance of the algorithm.

8.3. δ selection

We have analyzed the effect of selection of δ_j in Equation (2) and δ_{a_j} in Equation (6) using the benchmark dataset *Corrected* KDD Cup 1999 and two real-life datasets, viz., *Packet level* and *Flow level* TUIDS datasets. The performance of the proposed method in terms of *recall* depends on the selection of δ_j and δ_{a_j} as seen in Figs 5 and 6. It is dependent on the dataset used for evaluation. However, a most probable range of δ_j and δ_{a_j} for these datasets is shown with vertically drawn dashed lines in Figs 5 and 6. In our experiments, better results are obtained with $\delta_j = 0.6$ and $\delta_{a_j} = 0.65$.

TABLE 11. Packet level features of the TUIDS intrusion dataset.

Sl.	Feature names	Type*	Feature description
Basic features			
1.	Duration	C	Time since occurrence of the first frame
2.	Protocol	D	Protocol of layer 3- IP, TCP, UDP
3.	Src IP	C	Source IP address
4.	Dst IP	C	Destination IP address
5.	Src port	C	Source port of machine
6.	Dst port	C	Destination port of machine
7.	Service	D	Network service on the destination e.g. http, telnet, etc.
8.	num-bytes-src-dst	C	No. of data bytes flowing from src to dst
9.	num-bytes-dst-src	C	No. of data bytes flowing from dst to src
10.	Fr-no.	C	Frame number
11.	Fr-length	C	Length of the frame
12.	Cap-length	C	Captured frame length
13.	Head-len	C	Header length of the packet
14.	Frag-offset	D	Fragment offset value
15.	TTL	C	Time to live
16.	Seq-no.	C	Sequence number
17.	CWR	D	Congestion window record
18.	ECN	D	Explicit congestion notification
19.	URG	D	Urgent TCP flag
20.	ACK	D	Ack flag
21.	PSH	D	Push TCP flag
22.	RST	D	Reset RST flag
23.	SYN	D	Syn TCP flag
24.	FIN	D	Fin TCP flag
25.	Land	D	1 if connection is from/to the same host/port; 0 otherwise
Content-based features			
26.	Mss-src-dst-requested	C	Maximum segment size from src to dst requested
27.	Mss-dst-src-requested	C	Maximum segment size from dst to src requested
28.	Ttt-len-src-dst	C	Time to live length from src to dst
29.	Ttt-len-dst-src	C	Time to live length from dst to src
30.	Conn-status	C	Status of the connection (1-complete, 0-reset)
Time-based features			
31.	count-fr-dst	C	No. of frames received by the unique dst in the last T sec from the same src
32.	count-fr-src	C	No. of frames received by the unique src in the last T sec to the same dst
33.	count-serv-src	C	No. of frames from the src to the same dst port in the last T sec
34.	count-serv-dst	C	No. of frames from dst to the same src port in the last T sec
35.	num-pushed-src-dst	C	No. of pushed pkts flowing from src to dst
36.	num-pushed-dst-src	C	No. of pushed pkts flowing from dst to src
37.	num-SYN-FIN-src-dst	C	No. of SYN/FIN pkts flowing from src to dst
38.	num-SYN-FIN-dst-src	C	No. of SYN/FIN pkts flowing from dst to src
39.	num-FIN-src-dst	C	No. of FIN pkts flowing from src to dst
40.	num-FIN-dst-src	C	No. of FIN pkts flowing from dst to src
Connection-based features			
41.	count-dst-conn	C	No. of frames to the unique dst in the last N packets from the same src
42.	count-src-conn	C	No. of frames from the unique src in the last N packets to the same dst
43.	count-serv-src-conn	C	No. of frames from the src to the same dst port in the last N packets
44.	count-serv-dst-conn	C	No. of frames from the dst to the same src port in the last N packets
45.	num-packets-src-dst	C	No. of packets flowing from src to dst
46.	num-packets-dst-src	C	No. of packets flowing from dst to src
47.	num-acks-src-dst	C	No. of ack packets flowing from src to dst
48.	num-acks-dst-src	C	No. of ack packets flowing from dst to src
49.	num-retransmit-src-dst	C	No. of retransmitted packets flowing from src to dst
50.	num-retransmit-dst-src	C	No. of retransmitted packets flowing from dst to src

*C, continuous, D, discrete.

TABLE 12. Flow level features of the TUIDS Intrusion dataset.

Sl.	Feature names	Type*	Feature description
Basic features			
1.	Duration	C	Length of the flow (in seconds)
2.	Protocol-type	D	Type of protocols—TCP, UDP, ICMP
3.	src IP	C	Src node IP address
4.	dst IP	C	Destination IP address
5.	src port	C	Source port
6.	dst port	C	Destination port
7.	ToS	D	Type of service
8.	URG	D	Urgent flag of TCP header
9.	ACK	D	Ack flag
10.	PSH	D	Push flag
11.	RST	D	Reset flag
12.	SYN	D	SYN flag
13.	FIN	D	FIN flag
14.	Source byte	C	No. of data bytes transferred from the src IP addrs to the dst IP addrs
15.	dst byte	C	No. of data bytes transferred from the dst IP addrs to the src IP addrs
16.	Land	D	1 if connection is from/to the same host/port; 0 otherwise
Time-window features			
17.	count-dst	C	No. of flows to the unique dst IP addr inside the network in the last T sec from the same src
18.	count-src	C	No. of flows from the unique src IP addr inside the network in the last T sec to the same dst
19.	count-serv-src	C	No. of flows from the src IP to the same dst port in the last T sec
20.	count-serv-dst	C	No. of flows to the dst IP using the same src port in the last T sec
Connection-based features			
21.	count-dst-conn	C	No. of flows to the unique dst IP addrs in the last N flows from the same src
22.	count-src-conn	C	No. of flows from the unique src IP addrs in the last N flows to the same dst
23.	count-serv-src-conn	C	No. of flows from the src IP addrs to the same dst port in the last N flows.
24.	count-serv-dst-conn	C	No. of flows to the dst IP addrs to the same src port in the last N flows

*C, continuous; D, discrete.

TABLE 13. DDoS dataset.

	Categories				
Dataset	Normal	smurf	fraggle	synflood	Total
DDoS	71 599	9022	0	51 140	131 761
DDoS+	35 800	4510	6088	25 570	71 968

8.4. Experimental results

Experiments were performed using four datasets, viz., (i) Real-time TUIDS intrusion dataset, (ii) *DDoS Dataset*, (iii) KDD Cup 1999 dataset and (iv) NSL-KDD dataset were used to evaluate the performance of our method.

8.4.1. Results with TUIDS intrusion dataset

A. Packet Level dataset:

The confusion matrix for all-attacks categories on the *Packet Level* dataset is shown in Table 17. For cross-evaluation of the *Packet Level* dataset, *Packet Level* data is used for training and *Packet Level+* data is used for testing. The *Packet Level+* dataset has new attack records that do not exist in *Packet Level* dataset and these are marked as *Unknown* in our experiment. In cross-evaluation, training is carried out with training data and testing is performed with the test data. Cross-evaluation results are given in Table 18. In this case, a majority of *Unknown* attacks is misclassified as *DoS* and *Normal*. The average execution time of classification using cross-validation is 0.64 min.

B. Flow Level dataset:

The confusion matrix for all-attacks categories in the *Flow Level* datasets is shown in Table 19. The *Flow Level+* dataset has new attack records that do not exist in the *Flow Level dataset* and these are marked as *Unknown* in experiment. In cross evaluation training is carried out with training data and testing is performed with the test data. For cross-validation of the *Flow Level* dataset, *Flow Level* data is used for training and *Flow Level+* data is used for testing. Cross-validation results are shown in Table 20. The average execution time for classification using cross-validation is 0.25 min.

TABLE 14. Features of the *DDoS dataset*.

No.	Features	Type*	Description
1.	Duration	C	Length of the flow (in seconds)
2.	Protocol-type	D	Type of protocols—TCP, UDP, ICMP
3.	src IP	C	Source node IP address
4.	dest IP	C	Destination IP address
5.	src port	C	Source port
6.	dest port	C	Destination port
7.	ToS	D	Type of service
8.	URG	D	Urgent flag of TCP header
9.	ACK	D	Ack flag
10.	PSH	D	Push flag
11.	RST	D	Reset flag
12.	SYN	D	SYN flag
13.	FIN	D	FIN flag
14.	src byte	C	No. of data bytes transferred from the src IP addrs to the dst IP addrs
15.	Land	D	1 if connection is from/to the same host/port; 0 otherwise
16.	count-dst	C	No. of flows to the unique dst IP addrs inside the network in the last T sec (5 s) from the same src
17.	count-src	C	No. of flows from the unique src IP addrs inside the network in the last T sec (5 s) to the same dst
18.	count-serv-src	C	No. of flows from the src IP addrs to the same dst port in the last T sec (5 s)
19.	count-serv-dst	C	No. of flows to the dst IP addrs using the same src port in the last T sec (5 s)
20.	count-dst-conn	C	No. of flows to the unique dst IP addrs in the last N flows from the same src
21.	count-src-conn	C	No. of flows from the unique src IP addrs in the last N flows to the same dst
22.	count-serv-src-conn	C	No. of flows from the src IP addrs to the same dst port in the last N flows
23.	count-serv-dst-conn	C	No. of flows to the dst IP addrs to the same src port in the last N flows
24.	Label		Label of the feature instance as normal/attack

*C, continuous; D, discrete.

TABLE 15. Attacks distribution in KDD Cup datasets.

Datasets	DoS	R2L	U2R	Probe	Normal	Total
Corrected KDD	229 853	16 347	70	4166	60 593	311 029
10-percent KDD	391 458	1126	52	4107	97 278	494 021

TABLE 16. NSL-KDD dataset.

Datasets	DoS	U2R	R2L	Probe	Normal	Total
$KDDTrain^+$	45 927	52	995	11 656	67 343	125 973
$KDDTest^+$	7458	67	2887	2422	9710	22 544

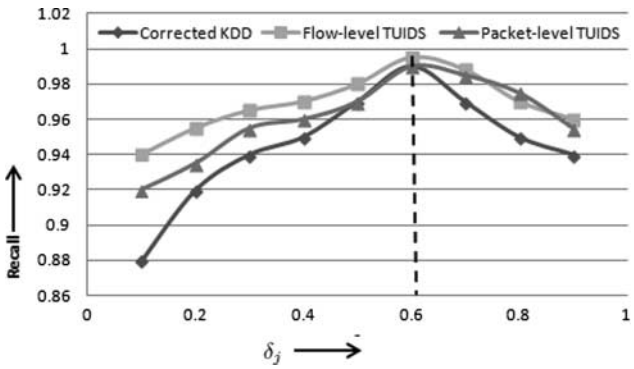

FIGURE 5. δ_j selection.

8.4.2. Results using the DDoS dataset

The confusion matrix for all-attacks categories on the *DDoS* is shown in Table 21. The *DDoS+* dataset has new attack records that do not exist in the *DDoS* dataset and these are marked as *Unknown* in experiment. In cross evaluation, training is carried out with training data and testing is performed with the test data. Cross evaluation results are given in Table 22. In this case, a majority of *Unknown* attacks is misclassified as *DoS* and *Normal*. The average execution time of classification using cross evaluation is 0.21 min.

8.4.3. Results using KDD Cup 1999 dataset

The confusion matrices for KDD Cup 1999 dataset on the *Corrected KDD* and the 10% *KDD* datasets are shown in Tables 23 and 24, respectively. Cross-validation results are given in Table 25. In cross validation, the 10-*percent KDD*

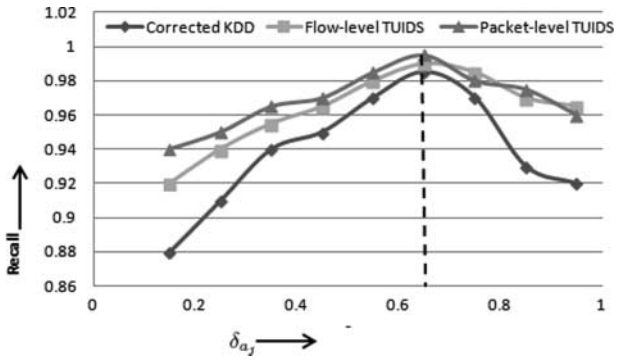

FIGURE 6. δ_{a_j} selection.

TABLE 17. Confusion matrix for *Packet Level* dataset.

Actual class	Predicted class DoS	Probe	Normal	Sum	Recall	1-Prc*
DoS	150 885	103	215	151 203	0.9979	0.0148
Probe	86	26 557	196	26 839	0.9895	0.0176
Normal	2380	373	252 080	254 833	0.9892	0.0016
Sum	153 151	27 033	252 491	432 875		

*1-Precision.

TABLE 18. Confusion matrix for cross-evaluation with *Packet Level* data for training and *Packet Level*+ data for testing.

Actual class	Predicted class DoS	Probe	Normal	Un@	Sum	Recall	1-Prc*
DoS	71 910	161	303	226	72 600	0.9905	0.0519
Probe	134	10 075	103	578	10 890	0.9629	0.1165
Normal	3204	1162	121 656	1394	127 416	0.9548	0.0034
Un@	595	5	15	4915	5530	0.8888	0.3090
Sum	75 843	11 403	122 077	7113	216 436		

@Unknown, *1-Precision.

dataset is used for training and the *Corrected KDD* dataset is used for testing. The attack records that are present in the *Corrected KDD* dataset but do not exist in the 10-*percent KDD* dataset are marked as *Unknown*. In this case, the majority of *Unknown* attacks is misclassified as the *R2L* category. Average execution times for classification are 1.36, 1.91 and 1.42 min for *Corrected KDD*, the 10% *KDD* and for cross-validation, respectively.

8.4.4. *Results on NSL-KDD dataset*

The confusion matrices for the NSL-KDD dataset using the *KDDTrain*$^+$ and the *KDDTest*$^+$ are shown in Tables 26 and 27, respectively. For cross-validation, the *KDDTrain*$^+$ dataset is used for training and the *KDDTest*$^+$ dataset is used for testing. The attack records that exist in the *KDDTest*$^+$ dataset but do not exist in the *KDDTrain*$^+$ dataset are marked as *Unknown*. Cross-validation results are given in Table 28. In this case, the

TABLE 19. Confusion matrix for *Flow Level* dataset.

Actual class	Predicted class DoS	Probe	Normal	Sum	Recall	1-Prc*
DoS	162 089	52	192	162 333	0.9985	0.0105
Probe	146	62 057	538	62 741	0.9891	0.0137
Normal	1569	812	172 676	175 057	0.9864	0.0042
Sum	163 804	62 921	173 406	400 131		

*1-Precision.

TABLE 20. Confusion matrix for cross evaluation with *Flow Level* data for training and *Flow Level*+ data for testing.

Actual class	Predicted class DoS	Probe	Normal	Un@	Sum	Recall	1-Prc*
DoS	77 840	153	194	479	78 666	0.9895	0.0317
Probe	165	29 384	170	656	30 375	0.9674	0.0439
Normal	2298	1147	82 407	1676	87 528	0.9415	0.0070
Un@	84	48	220	3148	3500	0.8994	0.4717
Sum	80 387	30 732	82 991	5959	200 069		

@Unknown, *1-Precision.

TABLE 21. Confusion matrix for the *DDoS* dataset.

Actual class	Predicted class smurf	syn[a]	Normal	Sum	Recall	1-Prc*
smurf	8869	43	110	9022	0.9831	0.0503
syn[a]	316	50 096	728	51 140	0.9796	0.0028
Normal	154	97	71 348	71 599	0.9965	0.0116
Sum	9339	50 236	72 186	131 761		

[a]synflood, *1-Precision.

TABLE 22. Confusion matrix for cross evaluation with *DDoS* data for training and *DDoS*+ data for testing.

Actual class	Predicted class smurf	syn[a]	Normal	fra[b]	Sum	Recall	1-Prc*
smurf	4421	24	26	39	4510	0.9804	0.0935
syn[f]	219	24 749	328	274	25 570	0.9679	0.0108
Normal	51	97	35 427	225	35 800	0.9896	0.0190
fra[g]	186	150	331	5421	6088	0.8905	0.0903
Sum	4877	25 020	36 112	5959	71 968		

[a]synflood, [b]fraggle, *1-Precision.

TABLE 23. Confusion matrix of the *Corrected KDD* dataset.

	Predicted class							
Actual class	DoS	Probe	Normal	U2R	R2L	Sum	Recall	1-Prc*
DoS	229 828	2	23	0	0	229 853	0.9999	0.0001
Probe	7	4114	42	0	3	4166	0.9875	0.0044
Normal	6	15	54 574	1	5997	60 593	0.9007	0.0273
U2R	0	0	10	57	3	70	0.8143	0.0272
R2L	0	1	1452	2	14 892	16 347	0.9110	0.2873
Sum	229 841	4132	56 101	60	20 895	311 029		

*1-Precision.

TABLE 24. Confusion matrix of the 10-*percent KDD* dataset.

	Predicted class							
Actual class	DoS	Probe	Normal	U2R	R2L	Sum	Recall	1-Prc*
DoS	391 450	1	7	0	0	391 458	1.0000	0.0000
Probe	0	4092	15	0	0	4107	0.9963	0.0104
Normal	1	42	97 221	0	14	97 278	0.9994	0.0012
U2R	0	0	6	44	2	52	0.8462	0.0435
R2L	0	0	88	2	1036	1126	0.9201	0.0152
Sum	391 451	4135	97 337	46	1052	494 021		

*1-Precision.

TABLE 25. Confusion matrix for cross evaluation of MLH-IDS with 10-*percent KDD* for training and *Corrected KDD* for testing.

	Predicted class								
Actual class	DoS	Probe	Normal	U2R	R2L	Unknown	Sum	Recall	1-Precision
DoS	221 903	102	702	19	42	530	223 298	0.9938	0.0047
Probe	43	2247	22	4	10	51	2377	0.9453	0.2439
Normal	433	192	58 563	14	441	950	60 593	0.9665	0.0155
U2R	0	0	1	35	1	2	39	0.8974	0.6316
R2L	3	182	170	5	5273	360	5993	0.8798	0.2620
Unknown	565	249	26	18	1378	16 493	18 729	0.8806	0.1030
Sum	222 947	2972	59 484	95	7145	18 386	311 029		

majority of *Unknown* attacks is misclassified as *DoS*, *Probe* and *R2L* categories. The average execution times for classification are 0.67, 0.11 and 0.17 min for $KDDTrain^+$ dataset, $KDDTest^+$ dataset and for cross evaluation, respectively.

8.5. Comparison of results

We performed experiments using the TUIDS datasets, viz., the *Packet level*, and *Flow level* and *DDoS* datasets using the decision tree-based algorithm *C*4.5 [51] and a graph-based *Bayesian Networks* algorithm following [52] and the *Naive Bayes* algorithm [53] with Weka [54]. The results of comparison of performance using these algorithms and our method are given in Tables 29 and 30, respectively, for the TUIDS datasets at the *Packet level* and the *Flow level*. Experimental results of comparison using the above-mentioned algorithms and our method for the *DDoS* dataset are given in Table 31. The results of MLH-IDS outperforms the other three methods in all categories *DoS*, *Probe* and *Normal* for the *Packet level* and *Flow level* datasets. In case of the *DDoS*

TABLE 26. Confusion matrix of the *KDDTrain*$^+$ dataset.

	Predicted class							
Actual class	DoS	Probe	Normal	U2R	R2L	Sum	Recall	1-Prc*
DoS	45 900	4	23	0	0	45 927	0.9994	0.0025
Probe	0	11 563	93	0	0	11 656	0.9920	0.0173
Normal	114	199	66 883	5	142	67 343	0.9932	0.0026
U2R	0	0	9	43	0	52	0.8269	0.1224
R2L	0	0	47	1	947	995	0.9518	0.1304
Sum	46 014	11 766	67 055	49	1089	125 973		

*1-Precision.

TABLE 27. Confusion matrix of the *KDDTest*$^+$ dataset.

	Predicted class							
Actual class	DoS	Probe	Normal	U2R	R2L	Sum	Recall	1-Prc*
DoS	7443	0	14	0	1	7458	0.9980	0.0009
Probe	2	2399	16	0	5	2422	0.9905	0.0099
Normal	5	24	9556	0	125	9710	0.9841	0.0195
U2R	0	0	3	57	7	67	0.8507	0.0500
R2L	0	0	157	3	2727	2887	0.9446	0.0482
Sum	7450	2423	9746	60	2865	22 544		

*1-Precision.

TABLE 28. Confusion matrix for cross evaluation of MLH-IDS with the *KDDTrain*$^+$ dataset for training and the *KDDTest*$^+$ dataset for testing.

	Predicted class								
Actual class	DoS	Probe	Normal	U2R	R2L	Unknown	Sum	Recall	1-Precision
DoS	5680	0	14	0	1	46	5741	0.9894	0.0363
Probe	2	1072	16	0	5	11	1106	0.9693	0.1104
Normal	5	24	9515	0	89	77	9710	0.9799	0.0156
U2R	0	0	3	31	2	1	37	0.8378	0.0723
R2L	0	0	79	3	1913	179	2174	0.8799	0.0385
Unknown	207	109	39	9	52	3360	3776	0.8898	0.0855
Sum	5894	1205	9666	43	2062	3674	22 544		

dataset, MLH-IDS outperforms the other methods for *Normal* and *synflood* categories.

A summarized comparison of results of MLH-IDS with other competing methods reported in [51, 55–57] for KDD Cup 1999 intrusion datasets is given in Table 32. As seen in Table 32, MLH-IDS outperforms all other methods for DoS, Probe and R2L categories. In case of U2R category attack, except the method reported in [55], MLH-IDS outperforms all other methods. Work is underway to improve the performance of the k-point algorithm with cluster stability analysis for more accurate identification of normal instances.

9. CONCLUSIONS AND FUTURE WORK

The work presented in this paper proposes a multi-level hybrid intrusion detection method based on supervised, unsupervised and outlier methods. The proposed method exhibits very good performance in detecting rare category attacks as well as

TABLE 29. Comparison of results for the TUIDS dataset *Packet level*.

Category	Measure	Baysian network [52]	Naive Bayes [53]	C4.5 [51]	MLH-IDS
DoS	*Recall*	0.9911	0.9891	0.9807	0.9979
	*1-Prc**	0.0230	0.0219	0.0124	0.0148
Probe	*Recall*	0.9376	0.8580	0.9530	0.9895
	*1-Prc**	0.0212	0.0084	0.0168	0.0176
Normal	*Recall*	0.9750	0.9240	0.9820	0.9892
	*1-Prc**	0.0106	0.0145	0.0152	0.0016

*1-Precision.

TABLE 30. Comparison of results for the TUIDS dataset *Flow level*.

Category	Measure	Baysian network [52]	Naive Bayes [53]	C4.5 [51]	MLH-IDS
DoS	*Recall*	0.9760	0.8540	0.9860	0.9985
	*1-Prc**	0.0190	0.0860	0.0140	0.0105
Probe	*Recall*	0.9890	0.9040	0.9850	0.9891
	*1-Prc**	0.0050	0.0814	0.0030	0.0137
Normal	*Recall*	0.9550	0.2690	0.9750	0.9864
	*1-Prc**	0.0830	0.0120	0.0480	0.0042

*1-Precision.

TABLE 31. Comparison of results for the *DDoS Dataset*.

Category	Measure	Baysian network [52]	Naive Bayes [53]	C4.5 [51]	MLH-IDS
Normal	*Recall*	0.9730	0.4520	0.9920	0.9965
	*1-Prc**	0.0030	0.0140	0.0240	0.0116
smurf	*Recall*	1.0000	0.9570	0.9810	0.9831
	*1-Prc**	0.5580	0.2670	0.0061	0.0503
synflood	*Recall*	0.9795	0.9705	0.9730	0.9796
	*1-Prc**	0.7530	0.5070	0.0048	0.0028

*1-Precision.

TABLE 32. Comparison of detection rates for the *Corrected KDD* dataset (%).

Category	Three-level tree classifier [55]	Multiple-level hybrid classifier [56]	SVM-based IDS [57]	C4.5 [51]	MLH-IDS
DoS	98.54	99.19	99.53	99.99	99.99
Probe	93.50	98.71	97.55	94.82	98.75
Normal	94.68	96.80	99.29	94.42	90.07
U2R	97.14	66.67	19.73	67.11	81.43
R2L	48.91	89.14	28.81	81.53	91.10

large-scale attacks of both new and existing attacks when tested with several benchmark and real-life intrusion datasets. In further studies, we will strive to create a more effective ensemble approach based on faster and efficient classifiers so as to make a significant contribution in the study of the intrusion detection.

FUNDING

This work is an outcome of a research project on Network Security funded by DIT, MCIT, New Delhi.

REFERENCES

[1] Bace, R. and Mell, P. (2001) Intrusion Detection Systems. NIST Special Publications SP 800, U S Department of Defence, 31 November 2001.

[2] Lee, W. and Stolfo, S.J. (1998) Data Mining Approaches for Intrusion Detection. *Proc. 7th Conf. USENIX Security Symp.*, Vol. 7, San Antonio, TX, USA, January, pp. 6–6. USENIX.

[3] Roesch, M. (1999) Snort-Lightweight Intrusion Detection for Networks. *Proc. 13th USENIX Conf. System Administration*, Seattle, WA, USA, November, pp. 229–238. USENIX.

[4] Portnoy, L., Eskin, E. and Stolfo, S.J. (2001) Intrusion Detection with Unlabeled Data Using Clustering. *Proc. ACM CSS Workshop DMSA-2001*, Philadelphia PA, USA, November 8, pp. 5–8. ACM.

[5] Zhang, C., Zhang, G. and Sun, S. (2009) A Mixed Unsupervised Clustering-Based Intrusion Detection. *Proc. 3rd Int. Conf. Genetic and Evolutionary Computing, WGEC 2009*, Gulin, China, October 14–17. IEEE Computer Society.

[6] Kumar, S., Kumar, S. and Nandi, S. (2011) Multi-Density Clustering Algorithm for Anomaly Detection Using KDD'99 Dataset. *Proc. 1st Int. Conf. Advances in Computing and Communications (ACC 2011)*, Kochi, India, July 22–24, Communications in Computer and Information Science (CCIS) 190, pp. 619–630. Springer, Berlin.

[7] Gogoi, P., Bhattacharyya, D.K., Borah, B. and Kalita, J.K. (2011) A survey of outlier detection methods in network anomaly identification. *Comput. J.*, **54**, 570–588.

[8] Borah, B. and Bhattacharyya, D.K. (2008) Catsub: a technique for clustering categorical data based on subspace. *ICFAI J. Comput. Sci.*, **II**, 7–20.

[9] Gogoi, P., Borah, B. and Bhattacharyya, D.K. (2011) Network anomaly detection using unsupervised model. *Int. J. Comput. Appl. Spec. Issue Netw. Secur. Cryptogr.*, **NSC(1)**, 19–30.

[10] Lippmann, R. *et al.* (2000) Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation. *Proc. DARPA Information Survivability Conf. and Exposition (DISCEX) 2000*, Los Alamitos, CA, USA, pp. 12–26. IEEE Computer Society Press.

[11] Daniel, B., Julia, C., Sushil, J. and Ningning, W. (2001) ADAM: a testbed for exploring the use of data mining in intrusion detection. *SIGMOD Rec.*, **30**, 15–24.

[12] Burbeck, K. and Nadjm-Tehrani, S. (2005) ADWICE—Anomaly Detection with Real-Time Incremental Clustering. *Proc. Information Security and Cryptology—ICISC 2004*, Berlin, Germany, May, pp. 407–424. Springer, Berlin.

[13] Oke, G. and Loukas, G. (2007) A denial of service detector based on maximum likelihood detection and the random neural network. *Comput. J.*, **50**, 717–727.

[14] Garcia-Teodoro, P., Diaz-Verdejo, J., Macia-Fernandez, G. and Vazquez, E. (2009) Anomaly-based network intrusion detection: techniques, systems and challenges. *Comput. Secur.*, **28**, 18–28.

[15] Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A. and Stiller, B. (2010) An overview of IP flow-based intrusion detection. *IEEE Commun. Surv. Tutor.*, **12**, 343–356.

[16] Bhuyan, M.H., Bhattacharyya, D.K. and Kalita, J.K. (2011) Surveying port scans and their detection methodologies. *Comput. J.*, **54**, 1–17.

[17] Bhange, A. and Utareja, S. (2012) Anomaly detection and prevention in network traffic based on statistical approach and α-stable model. *Int. J. Adv. Res. Comput. Eng. Technol.*, **1**, 690–698.

[18] Chaturvedi, S.K., Richariya, V. and Tiwari, N. (2012) Anomaly detection in network using data mining techniques. *Int. J. Emerging Technol. Adv. Eng.*, **2**, 2250–2459.

[19] Zhang, T., Ramakrishnan, R. and Livny, M. (1996) Birch: An Efficient Data Clustering Method for Very Large Databases. *Proc. 1996 ACM SIGMOD*, Montreal, Quebec, Canada, June 4–6, pp. 103–114. ACM Press.

[20] Leung, K. and Leckie, C. (2005) Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters. *Proc. 28th Australasian Conf. Computer Science*, Vol. 38, Newcastle, NSW, Australia, January/February, pp. 333–342. Australian Computer Society, Inc. Darlinghurst.

[21] Casas, P., Mazel, J. and Owezarski, P. (2011) UNADA: Unsupervised Network Anomaly Detection Using Sub-Space Outliers Ranking. *Proc. 10th Int. IFIP TC 6 Conf. Networking—Volume Part I (NETWORKING'11)*, Heidelberg, Germany, pp. 40–51. Springer, Berlin.

[22] Wang, W., Yang, J. and Muntz, R.R. (1997) STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proc. of 23rd Int. Conf. on Very Large Databases*, San Francisco, CA, USA, pp. 186–195. Morgan Kaufmann.

[23] Nagesh, H.S., Goil, S. and Choudhary, A.N. (2000) A Scalable Parallel Subspace Clustering Algorithm for Massive Data Sets. *Proc. ICPP 2000*, Toronto, Canada, August 21–24, p. 477. IEEE Computer Society.

[24] Pan, Z.S., Chen, S.C., Hu, G.B. and Zhang, D.Q. (2003) Hybrid Neural Network and C4.5 for Misuse Detection. *Proc. 2003 Int. Conf. Machine Learning and Cybernetics*, Wan, pp. 2463–2467. IEEE Computer Society.

[25] Depren, O., Topllar, M., Anarim, E. and Ciliz, M.K. (2005) An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Syst. Appl.*, **29**, 713–722.

[26] Zhang, J. and Zulkernine, M. (2006) A Hybrid Network Intrusion Detection Technique Using Random Forests. *Proc. 1st Int. Conf. Availability, Reliability and Security, ARES 2006*, Vienna, Austria, pp. 262–269. IEEE Xplore.

[27] Hwang, K., Cai, M., Chen, Y. and Qin, M. (2007) Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. *IEEE Trans. Dependable Secur. Comput.*, **4**, 41–55.

[28] Aydyn, M.A., Zaim, A.H. and Ceylan, K.G. (2009) A hybrid intrusion detection system design for computer network security. *Comput. Electr. Eng.*, **35**, 517–526.

[29] Xiang, C., Chong, M.Y. and Zhu, H.L. (2004) Design of Multiple-Level Tree Classifiers for Intrusion Detection System. *Proc. 2004 IEEE Conf. Cybernetics and Intelligent Systems*, Singapore, December, pp. 872–877. IEEE Xplore.

[30] Amini, M., Jalili, R. and Shahriari, H.R. (2006) RT-UNNID: a practical solution to real-time network-based intrusion detection using unsupervised neural networks. *Comput. Secur.*, **25**, 459–468.

[31] Han, J. and Kamber, M. (2001) *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA.

[32] Kayacik, H.G., Heywood, A.N.Z. and Heywood, M.I. (2005) Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets. *Proc. 3rd Annual Conf. Privacy, Security and Trust*, Halifax, NS, Canada, October. Dalhousie University.

[33] Dunn, J. (1974) Well separated clusters and optimal fuzzy partitions. *J. Cybern.*, **4**, 95–104.

[34] Huberi, L. and Schultz, J. (1976) Quadratic assignment as a general data analysis strategy. *Br. J. Math. Stat. Psychol.*, **29**, 190–241.

[35] Davies, D.L. and Bouldin, D.W. (1979) A cluster separation measure. *IEEE Trans. Pattern Anal. Mach. Intell.*, **PAMI 1**, 224–227.

[36] Rousseeuw, P.J. (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, **20**, 53–65.

[37] Xie, X.L. and Beni, G. (1991) A validity measure for fuzzy clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, **13**, 841–847.

[38] Guo, Y. *et al*. (2010) Feature Selection Based on Rough Set and Modified Genetic Algorithm for Intrusion Detection. *Proc. 5th Int. Conf. Computer Science & Education*, Hefei, China, August 24–27, pp. 1441–1446. IEEEXplore.

[39] Jin, W., Tung, A.K.H., Han, J. and Wang, W. (2006) Ranking Outliers Using Symmetric Neighborhood Relationship. *Proc. 10th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD)*, Singapore, pp. 577–593.

[40] Katayama, N. and Satoh, S. (1997) The SR-Tree: An Index Structure for High-dimensional Nearest Neighbor Queries. *Proc. 1997 ACM SIGMOD Int. Conf. Management of data, SIGMOD '97*, NY, USA, June. ACM, New York.

[41] Gogoi, P., Bhuyan, M.H., Bhattacharyya, D.K. and Kalita, J.K. (2012) Packet and Flow Based Network Intrusion Dataset. *Proc. 5th Int. Conf. Contemporary Computing (IC3-2012)*, Noida, India, August 6–8, Communications in Computer and Information Science (CCIS) 306, pp. 322–334. Springer, Berlin.

[42] Hettich, S. and Bay, S.D. (1999) *The UCI KDD archive*. Irvine, CA: University of California, Department of Information and Computer Science. http://kdd.ics.uci.edu.

[43] Tavallaee, M., Bagheri, E., Lu, W. and Ghorbani, A.A. (2009) *A detailed analysis of the KDD CUP 99 data set*. http://nsl.cs.unb.ca/NSL-KDD/.

[44] Mixter, D. (2003) *Attacks tools and information*. http://packetstormsecurity.nl/index.html.

[45] Satten, C. (2008) *Lossless gigabit remote packet capture with Linux*. University of Washington Network Systems. http://staff.washington.edu/corey/gulp/.

[46] (2009). http://www.wireshark.org/.

[47] Quittek, J., Zseby, T., Claise, B. and Zender, S. (2004) *Rfc 3917: requirements for IP flow information export: IPFIX*, Hawthorn Victoria. http://www.ietf.org/rfc/rfc3917.txt.

[48] Claise, B. (2004) *Rfc 3954: Cisco Systems Netflow Services Export Version 9*. http://www.ietf.org/rfc/rfc3954.txt.

[49] Cisco.com (2010) *Cisco IOS NetFlow configuration guide, Release 12.4*. http://www.cisco.com.

[50] Haag, P. (2010) *Nfdump & Nfsen*. http://nfdump.sourceforge.net/.

[51] Quinlan, J.R. (1993) *C4.5:Programs for Machine Learning*. Morgan Kaufman.

[52] Jesen, F.V. (1996) *Introduction to Bayesian Networks*. UCL Press.

[53] Langley, P., Iba, W. and Thomas, K. (1992) An Analysis of Bayesian Classifiers. *Proc. 10th National Conf. Artificial Intelligence*, San Jose, CA, USA, July 12–16, pp. 223–228. AAAI Press.

[54] University of Waikato (1999) *Weka 3: Data Mining Software in Java*. http://www.cs.waikato.ac.nz/ml/weka.

[55] Lu, H. and Xu, J. (2009) Three-Level Hybrid Intrusion Detection System. *Proc. Int. Conf. Information Engineering and Computer Science, ICIECS 2009*, Shanghai, China, December 19–20, pp. 1–4. IEEE.

[56] Xiang, C., Yong, P.C. and Meng, L.S. (2008) Design of multiple-level hybrid classifier for intrusion detection system using Bayesian clustering and decision trees. *Pattern Recognit. Lett.*, **29**, 918–924.

[57] Horng, S.-J., Su, M.-Y., Chen, Y.-H., Kao, T.-W., Chen, R.-J., Lai, J.-L. and Perkasa, C.D. (2011) A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert Syst. Appl.*, **38**, 306–313.

[58] Dickerson, J.E. (2000) Fuzzy Network Profiling for Intrusion Detection. *Proc. 19th Int. Conf. North American Fuzzy Information Processing Society*, Atlanta, USA, July, pp. 301–306. IEEE.

[59] Labib, K. and Vemuri, R. (2002) NSOM: A Tool to Detect Denial of Service Attacks Using Self-Organizing Maps. Technical Report. Department of Applied Science University of California, Davis, CA, USA.

[60] Ertöz, L., Eilertson, E., Lazarevic, A., Ning Tan, P., Kumar, V. and Srivastava, J. (2004) The MINDS—Minnesota Intrusion Detection System. *Next Generation Data Mining*. MIT Press, Boston.

[61] Song, S., Ling, L. and Manikopoulo, C.N. (2006) Flow-Based Statistical Aggregation Schemes for Network Anomaly Detection. *Proc. IEEE Int. Conf. Networking, Sensing*, Florida, USA, April 23–25, pp. 786–791. IEEE.

[62] Mohajerani, M., Moeini, A. and Kianie, M. (2003) NFIDS: A Neuro-Fuzzy Intrusion Detection System. *Proc. 10th IEEE Int. Conf. Electronics, Circuits and Systems(ICECS)*, Iran, December, pp. 348–351. IEEE Xplore.

[63] Zhang, Z., Li, J., Manikopoulos, C.N., Jorgenson, J. and Ucles, J. (2001) HIDE: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification. *Proc. 2001 IEEE Man Systems and Cybernetics Information Assurance Workshop*, Seattle, WA, USA, April, pp. 85–90. IEEE Xplore.

[64] Sequeira, K. and Zaki, M. (2002) ADMIT: Anomaly-Based Data Mining for Intrusions. *Proc. 8th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, pp. 386–395. ACM Press.

[65] Subramoniam, N., Pawar, P.S., Bhatnagar, M., Khedekar, N.S., Guntupalli, S., Satyanarayana, N., Vijayakumar, V.A., Ampatt, P.K., Ranjan, R. and Pandit, P.S. (2005) Development of a Comprehensive Intrusion Detection System- Challenges and Approaches. *Proc. 1st Int. Conf. Information Systems Security (ICISS 2005)*, Kolkata, India, Lecture Notes in Computer Science 3803, pp. 332–335. Springer.

[66] Kuang, L.V. (2007) DNIDS: a dependable network intrusion detection system using the CSI-KNN algorithm. Master's Thesis, Queen's University Kingston, Ontario, Canada.

[67] Breunig, M.M., Kriegel, H.P., Ng, R.T. and Sander, J. (2000) LOF: Identifying Density-Based Local Outliers. *Proc. 2000 ACM SIGMOD Int. Conf. Management of Data*, Dallas, TX, USA, May 16–18, Vol. 29, pp. 93–104, ACM.

[68] Petrovskiy, M.I. (2003) Outlier detection algorithms in data mining systems. *Program. Comput. Softw.*, **29**, 228–237.

[69] Otey, M.E., Parthasarathy, S. and Ghoting, A. (2005) Fast Lightweight Outlier Detection in Mixed-attribute Data. Technical Report. Department of Computer Science and Engineering, The Ohio State University, Ohio, USA.

[70] Duan, L., Xu, L., Liu, Y. and Lee, J. (2008) Cluster-based outlier detection. *Ann. Oper. Res.*, **168**, 151–168.

[71] Lane, T. (2000) Machine learning techniques for the computer security domain of anomaly detection. PhD Thesis, Purdue University, Indiana, USA.

[72] Knorr, E.M. and Ng, R. (1998) Algorithms for Mining Distance-Based Outliers in Large Datasets. *Proc. 24th Int. Conf. Very Large Data Bases (VLDB 1998)*, New York, NY, USA, August 24–27, pp. 392–403. Morgan Kaufmann.