# CHAPTER 2 LITERATURE SURVEY AND BACKGROUND STUDY

## 2.1 Literature Survey

In this chapter, we present literature review consisting of two parts which are offline IDS and online IDS (real-time). The offline data uses KDD99 dataset which is a benchmark dataset. In this section, we focus on comparing various techniques for intrusion detection. Besides, we compare the performances of each technique, such as a detection rate, a false alarm rate and limitation. For the online IDS, we focus on a new technique to preprocess data in an actual network environment and testing environment.

### 2.1.1 IDS for Offline Data

Gómez and León [1] proposed a Fuzzy Genetic Algorithm to classify behavior of intrusion into two classes (normal class and attack class). This algorithm could be trained by one class (normal class). The behavior different from the training class would be classified as an attack. They used KDDC99 dataset which had four attack types including DoS, Probe, R2L and U2R. In the KDD99 dataset, they found that there were some features that had the same value for each record, so they reduced the number of the features into 33 features. The dataset was divided into two sets including the training set and the testing set. The training set had only the normal data containing 2,000 records. The highest obtained detection rate was 98.28% with 5% of the false alarm rate.

Banković et al. [2] proposed an interesting Fuzzy Genetic Algorithm Approach to reduce the number of the features in the dataset and maintain the high detection rate. From the experiment, they found that there were three features that were relevant. There were two experiments: the first experiment had two output classes (normal class and attack class). The accuracy of the detecting attack (TN) was 94.87% with 1.62% of the false positive. The second experiment had four classes (the fuzzy rule could identify each type of the attacks including the normal class, the *portsweep* class, the *smurf* class and the *neptune* class). From this experiment, the maximum detection rate was 87.6% because there was only 30% of the detection rate of the *portsweep*. These two experiments used the KDD99 dataset. However, the training dataset had 976 records (137 of attack records and 839 of normal records) and the testing dataset had 977 records (234 of attack records and 743 of normal records). Moreover, they considered only three types of the attacks which were the *portsweep*, *smurf* and *neptune*.

Tsang et al. [3] proposed Multi-Objective Genetic Fuzzy Intrusion Detection System (MOGFIDS) for detecting anomaly attack. There were three objectives for MOGFIDS: having the high classification rate, reducing the number of fuzzy rules and reducing complexity of fuzzy rules. This experiment used 10% version of KDD99 dataset for training including four attack types (DoS, Probe, R2L and U2R). However, they found that the dataset was biased against DoS (Neptune attack and Smurf attack). In order to make the training set more realistic, they sampled 1,000 records for each type of the DoS, 10,000 records of the normal and the remaining intact number of the records of other attacks (the number of the training set was 20,752 records). The testing set used 311,029 records with additional 14 unseen attack types. The result showed that this algorithm with 27 features gave 92.77% of the detection rate and 1.6 of the false positive rate.

Ensafi et al. [4] proposed optimizing fuzzy K-means for network anomaly detection using particle swarm optimization (PSO). Two versions of the KDD99 dataset were used (full version and 10% version). The training dataset had only the normal class from the 10% version. The testing dataset consisted of 60,592 records of the normal class and 250,436 of the attack class. Figure 2.1 presents the diagram of the proposed work. Particles swarm and K-means clustering was used together to cluster the dataset in each generation. A genetic algorithm was used to find the best solution. The output classes were Normal, DoS, R2L, U2R and Probe, and the detection rate was 95 % with 2.12% of the false alarm rate.

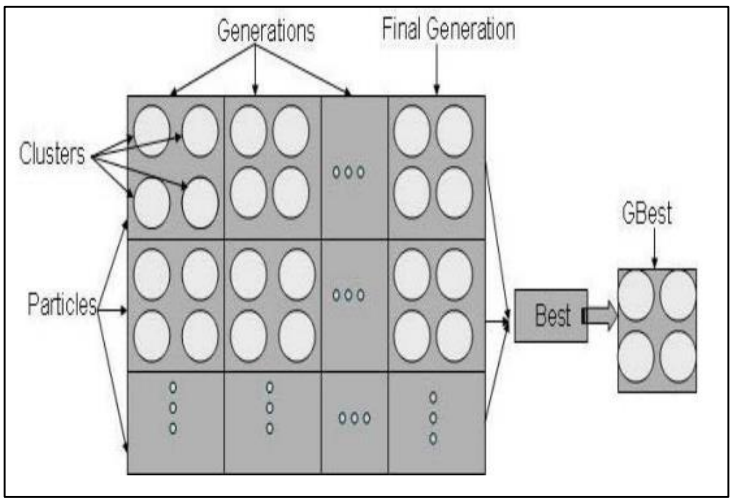



**Figure 2.1** Optimizing fuzzy K-means for network anomaly framework [4]

Fries [5] proposed a Fuzzy Genetic Algorithm Approach. This work had two phases: preprocessing phase and detection phase. In the preprocessing phase, they used clustering and genetic algorithm to find the significant features. The result showed that there were 8 relevant features. In the detection phase, they evaluated the algorithm by using the 10% version of the KDD99 dataset as the training set (about 500,000 records) and the full version of the KDD99 dataset as the testing set (about 5 million records). In the testing set, there were 14 types of new attacks that were not presented in the training set. The detection rate was 99.6% with 0.2 of the false positive rate. They found that this algorithm had the high detection rate and was robust for an untrained attack.

Abadeh et al. [6] proposed a genetic fuzzy algorithm. They used three different kinds of genetic fuzzy systems based on Michigan, Pittsburgh and iterative rule learning. The algorithm could be classified into five classes (Normal, U2R, R2L, DoS, and Probe). The distribution of the training dataset and the testing dataset is shown in Table 2.1. The result showed that the Pittsburgh method had the highest detection rate of 99.53% with 1.94% of the false alarm rate.

**Table 2.1** Distribution of different classes in training and testing datasets [6]

| Attack Type | Train | Test |
|---|---|---|
| Normal | 100 | 1,000 |
| U2R | 50 | 59 |
| R2L | 100 | 1,000 |
| DoS | 300 | 6,500 |
| PORBE | 100 | 1,000 |
| **Total** | **650** | **9,559** |

Ngamwitthayanon and Wattanapongsakorn [7] proposed a Fuzzy-Adaptive Resonance Theory (ART) in network anomaly detection with feature-reduction dataset. The Adaptive Resonance was a type of the neural network algorithm. The main algorithm was the ART algorithm while the Fuzzy was used to simplify a network structure of the ART. Moreover, they applied a feature reduction method with the KDD99 dataset. This approach increased the detection rate to 98.96% and used only14 features. However, this algorithm indicating the similar problem as the previous algorithm was impractical in the real network. Also, it did not provide enough information for a protection system.

**Table 2.2** Detection rate with different numbers of KDD99 features [7]

| Dataset | Number of Features | Detection Rate (%) |
|---|---|---|
| 1 | 7 | 98.87 |
| 2 | 9 | 99.44 |
| 3 | 12 | 98.98 |
| 4 | 14 | 98.93 |
| 5 | 22 | 99.12 |
| 6 | 24 | 99.20 |
| 7 | 41 | 97.96 |

Muda et al. [8] proposed a detection solution by combining of the K-means algorithm and the Naïve Bayes algorithm. The first step of the algorithm was using the K-means algorithm to categorize data into two classes; normal class and attack class. Then, the Naïve Bayes algorithm was used to classify the previous results into attack types. They sampled 49,402 records of the training set from the 10% version of the KDD99 dataset and another 49,402 records from the full version of the KDD99 which had more 14 types of new attacks. The detection rate was 99.6%. However, this solution was impractical for a real network environment because the K-Means algorithm required time to process. It could cause the bottleneck problem in network traffic or system clash.

Seungmin et al. [9] proposed a self-organizing map (SOM) combined with the K-means algorithm to classify untrained attacks. The system was able to learn from the new data. There were three phases consisting of an adjusting SOM network, updating centroid (K-means algorithm) and splitting normal cluster. The cluster system could divide the output into two classes (normal class and attack class). They sampled the dataset from the KDD99 dataset. The size of the sampling dataset was 20,000 records which consisted of 1% of the attack and 99% of the normal class. They reduced the number of features into

eight features (2, 3, 4, 10, 12, 23, 33 and 35). The average detection rate in this work was 89.7% with 2.43 of the false positive rate.

Chandrasekhar and Raghuveer [10] proposed an intrusion detection technique using the K-means, fuzzy neural network and the SVM algorithm. They found that a rule based system was worse when encountering with a large scale of the data, so they introduced the artificial neural network (ANN) for this system [Figure 2.2]. First, they used the K-mean algorithm to cluster the dataset into n clusters (each cluster was the type of intrusion). In each cluster, there was a neuro-fuzzy to learn the pattern. The neuro-fuzzy in each cluster was used to generate the SVM vector to classify attacks (the neuro-fuzzy algorithm helped to decrease a number of attributes in SVM). They sampled the training dataset and testing dataset from a 10% version file of the KDD99 dataset which consisted of 26,114 records for the training dataset and 27,112 records for the testing dataset (Table 2.3). The accuracy of each attack was 98% for DoS attack, 97.31% for Probe, 97.51 for R2L and 97.52 for U2R. Total detection rate was 98.48% with 2.41 % of the false positive rate.

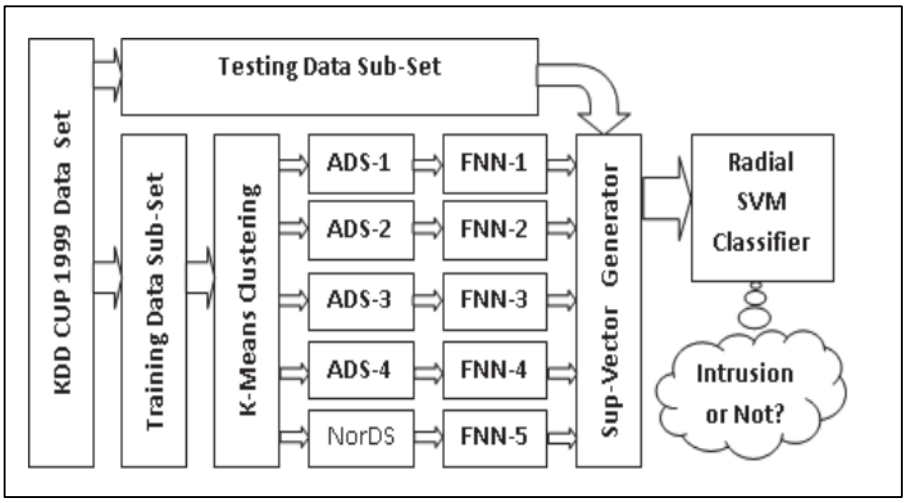



**Figure2.2** Block diagram of proposed IDS from using K-means, fuzzy neural network and SVM algorithm [10]

**Table 2.3** Data record taken for training and testing in [10]

| | **Normal** | **DoS** | **PROBE** | **R2L** | **U2R** | **TOTAL** |
|---|---|---|---|---|---|---|
| **Training** | 12,500 | 12,500 | 1,054 | 39 | 21 | 26,114 |
| **Testing** | 12,500 | 12,500 | 2,053 | 38 | 21 | 27,112 |

**Table 2.4** Summary of Offline IDS

| Year | Author | Algorithm | DR(%) | FP(%) | Feature | Output |
|---|---|---|---|---|---|---|
| 2006 [1] | Gómez and León | Genetic Fuzzy | 98.28 | N/A | 33 | Normal, Attack |
| 2007 [2] | Banković et al | Genetic Fuzzy | 94.87 | 0-1.62 | 3 | Normal, Attack |
| | | | 87.6 | 0 | | Normal, Portsweep, Smurf and Neptune |
| 2007 [3] | Tsang et al. | Genetic Fuzzy | 92.77 | 1.6 | 27 | Normal, Probe, DoS, U2R, R2L |
| 2008 [4] | Ensafi et al | Fuzzy K-means and PSO | 95.9 | 2.12 | 33 | Normal, Probe, DoS, U2R, R2L |
| 20010 [5] | Fries | Genetic Fuzzy | 99.6 | 0.2 | 8 | Normal, attack |
| 2010 [6] | Abadeh et al. | Genetic Fuzzy | 99.53 | 1.94 | 21 | Normal, Probe, DoS, U2R, R2L |
| 2011 [7] | Ngamwitthayanon and Wattanapongsakorn | Fuzzy and ART | 98.96 | N/A | 14 | Normal, Attack |
| 2011 [8] | Muda et al. | K-means+ naïve bayes technique | 99.8 | 0.09 | 41 | Normal, Probe, DoS, U2R, R2L |
| 2011 [9] | Seungmin et al. | SOM and K-means | 89.7 | 2.43 | 8 | Normal, attack |
| 2013 [10] | Chandrasekhar et al. | K-means, fuzzy neural network and SVM | 98.48 | 2.41 | N/A | Normal, Probe, DoS, U2R, R2L |

** DR = Detection Rate
** FA = False Alarm
** N/A not available

### 2.1.2 IDS for Online Data

Labib and Vemuri [11] proposed a real-time intrusion detection system by considering 10 features of header packets. Each record was the statistic data which was collected in every 50 packets. Then, they used SOM as an algorithm to classify attacks. The outputs were normal and DoS attacks. On the other hand, it needed a human expert to visualize the output data.

Amini et al. [12] proposed a real-time intrusion detection system using neural network algorithms (Adaptive Resonance Theory (ART) and Self-Organizing Map (SOM)) to classify normal packets and attack packets (two classes) as shown in Figure 2.3. They generated the attacks and collected the attack data by using attack tools as shown in Table 2.5 (left). They collected normal traffic in a real traffic network within 4 days. So, they created their own dataset which consisted of training data (5,000 packets) and testing dataset (3,000 packets). They preprocessed the packets into 27 features as shown in Table 2.5 (right). The result showed that the ART had the higher detection of 97.42%. The result is shown in Table 2.5.

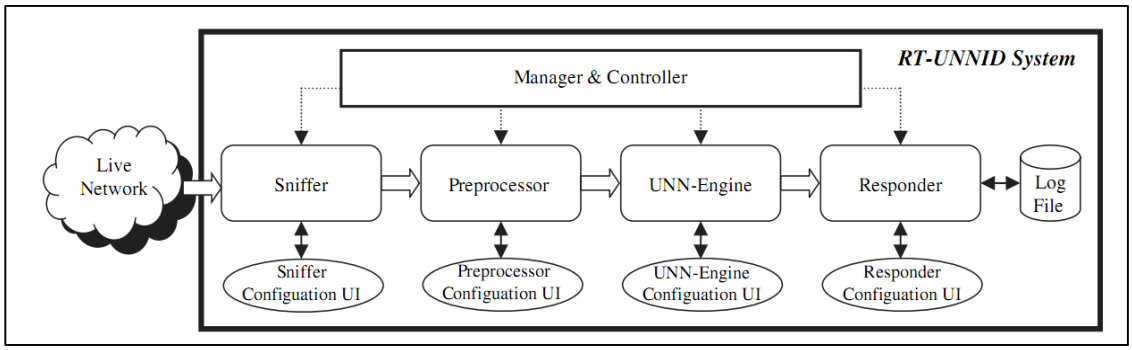



**Figure 2.3** RT-UNNID systems [12]

**Table 2.5** Real-time detection rate of RT-UNNID using SOM ART-1 and ART-2 [12]

| | **ETTR** | **TR** | **FPR** | **FNR** |
|---|---|---|---|---|
| ART-1 | 71.71 | 97.42 | 1.99 | 0.59 |
| ART-2 | 73.18 | 97.19 | 2.3 | 0.51 |
| SOM | 83.44 | 95.74 | 3.5 | 0.77 |

**ETTR is exact true types detection rate
TR is true detection rate
FPR is false positive detection rate
NFR is false negative detection rate

**Table 2.6** Attack name (left) and feature name in proposed approach (right) [12]

| # | Attack name | Attack generation tools | Train dataset | Test dataset |
|---|---|---|---|---|
| 1 | Bonk | targa2.c | √ | √ |
| 2 | Jolt | targa2.c | √ | √ |
| 3 | Land | targa2.c | √ | √ |
| 4 | Saihyousen | targa2.c | √ | √ |
| 5 | TearDrop | targa2.c | √ | √ |
| 6 | Newtear | targa2.c | √ | √ |
| 7 | 1234 | targa2.c | √ | √ |
| 8 | Winnuke | targa2.c | √ | √ |
| 9 | Oshare | targa2.c | √ | √ |
| 10 | Nestea | targa2.c | √ | √ |
| 11 | SynDrop | targa2.c | √ | √ |
| 12 | Octopus | Octopus.c | √ | √ |
| 13 | KillWin | KillWin.c | √ | √ |
| 14 | Twinge | Twinge.c | √ | √ |
| 15 | TcpWindowScan | Nmap | √ | √ |
| 16 | SynScan | Nmap | √ | √ |
| 17 | Neptune | FireHack | √ | √ |
| 18 | Dosnuke | FireHack | √ | √ |
| 19 | Smbdie | Smbdie.exe | √ | √ |
| 20 | XmassTree-Scan | Namp | √ | √ |
| 21 | LinuxICMP | linux-icmp.c | - | √ |
| 22 | Moyari13 | Moyari13.c | - | √ |
| 23 | Sesquipedalian.c | Sesquipedalian | - | √ |
| 24 | Smurf | smurf4.c | - | √ |
| 25 | OverDrop | overdrop.c | - | √ |
| 26 | OpenTear | opentear.c | - | √ |
| 27 | ExhoChargen | FireHack | - | √ |

| Category | Feature |
|---|---|
| - | protocol |
| **IP** | diff-time stamp |
| | ip id |
| | IP tos |
| | ipttl |
| | ipheaderlen |
| | iplen |
| | is home srcip |
| | is home dstip |
| | is land |
| | ip frag flag |
| **TCP** | tcpsrc port |
| | tcpdst port |
| | tcp fin |
| | tcpsyn |
| | tcprst |
| | tcp push |
| | tcpack |
| | tcpurg |
| | tcp offset |
| | tcp win size |
| **UDP** | udpsrc port |
| | udpdst port |
| **ICMP** | icmp type |
| | icmp code |
| | icmp id |
| | icmp sequence |

Pukkawanna et al. [13] proposed the Lightweight Detection system ($LD^2$) to detect Denial of Service Attack (DoS). The target attacks included SYN Flood, ICMP flood, Port scan Host scan, UDP flood and smurf. The system preprocessed the network into five features (srcIP, protocol, dstIP, srcPort, and dstPort). The background traffic environment had two types: controlled environment and real traffic environment. In the controlled network environment, they used Iperf to generate the UDP traffic in various rates. In the real network environment, they replied traces by using tcpreplay. The trace was sampled from WIDE Backbone (100-150 Mbps). In each experiment, they generated a DoS attack on the top of a single background trace. Figure 2.4 showed the graph pattern that the system used for detecting each type of the attacks. For example, SYN flood had the same (srcIP, prot, dstIP, dstPort) but various srcPort. Thus, the detection system needed the training process in order to find a threshold for each attack type (Table 2.7). They generated multiple attacks at once (12 instances). The experiment result showed that the $LD^2$

performed well with the 100% detection rate (except a host scan that could not detect some activities) with no false positive. They also evaluated a system performance in term of CPU consumption and memory consumption by using a systat tool. It showed that the increasing packet rate of a background also increased the CPU usage [Figure 2.5]. The maximum CPU utilization of the $LD^2$ was 16% at 7,000 pps and the memory consumption was 20 MB. The behavior of the memory consumption is shown in [Figure 2.6].

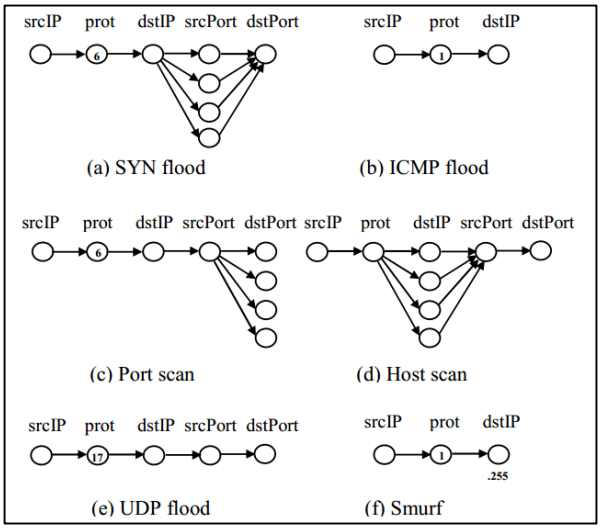



**Figure 2.4** DoS attack graphlets [13]

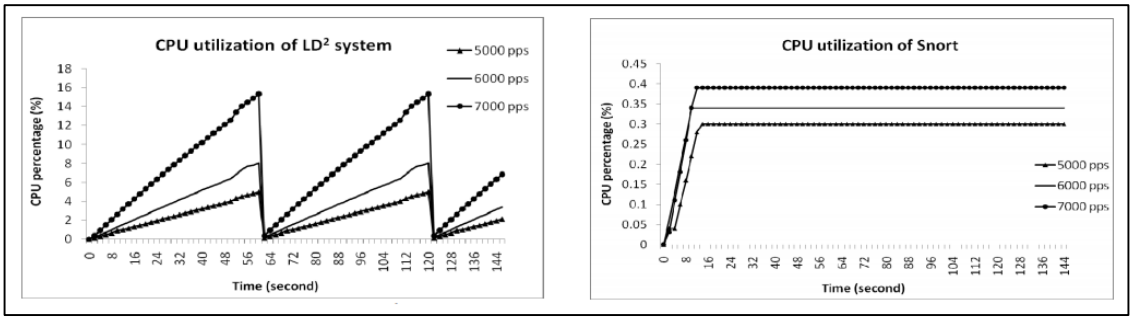



**Figure 2.5** CPU initialized for $LD^2$ (left) and Snort (right) [13]

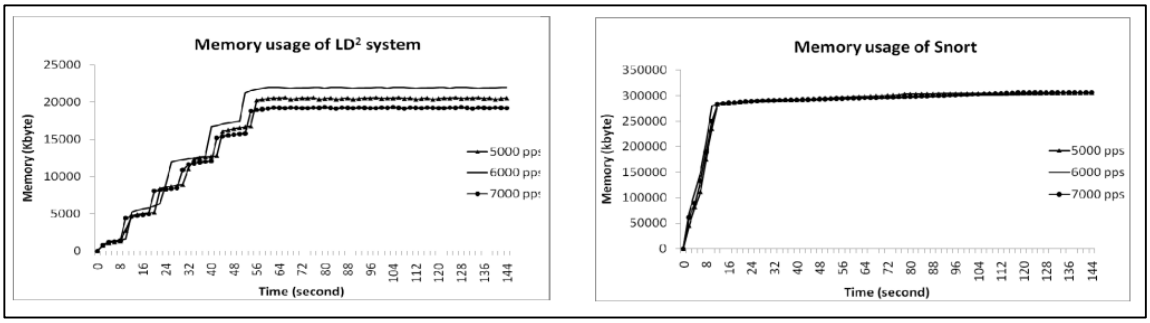



**Figure 2.6** Memory usage for $LD^2$ (left) and Snort (right) [13]

**Table 2.7** Threshold for attack graphlets [13]

| Dos Type | Threshold Parameters (per minute) | Upper Bound | Suggest Value |
|---|---|---|---|
| SYN flood | Source ports | 1,998 | 1,598 |
| UDP flood | Number of UDP packets | 1,918 | 1,534 |
| ICMP flood | Number of ICMP packets to broadcast address | 2,151 | 1,721 |
| Smurf | Number of ICMP packets to broadcast address | 2,151 | 1,721 |
| Port scan | Destination ports | 394 | 313 |
| Host scan | Destination IP adresses | 5 | 4 |

Su [14] proposed the real-time IDS for large-scale attacks by using fuzzy association rules. The technique derived features from a packet header from the open network within every 2 seconds (one record per two seconds). There were 16 features used in this technique as shown in Table 2.8. The system architecture is shown in Figure 2.7. The computer A preprocessed data from a real network and sent a record to the computer B to create a fuzzy rule. The computer D compared the rules between the computer B and C to find the attacks. This experiment was tested on 30 DoS attacks. A network topology is shown in Figure 2.8. IP traffic (a sender) was a computer used to generate the background traffic, such as TCP packets, UDP packets, ICMP packets and ARP packets. It connected to the internet. There was the IP traffic (a receiver) located in the local network. An attack generator was used to generate attacks (DoS) where the victim was found in the local network. The system was also located in the local network. It monitored the traffic in the local network. The traffic rate during the experiment was 0-80 Mbps. The result is shown in Figure 2.9. We can see that the system responded to the attack five time units (10 seconds) after the system was attacked. This system could only give an alarm signal when the network was under attack. However, it could not provide any useful information to prevent the network from malicious network activities.

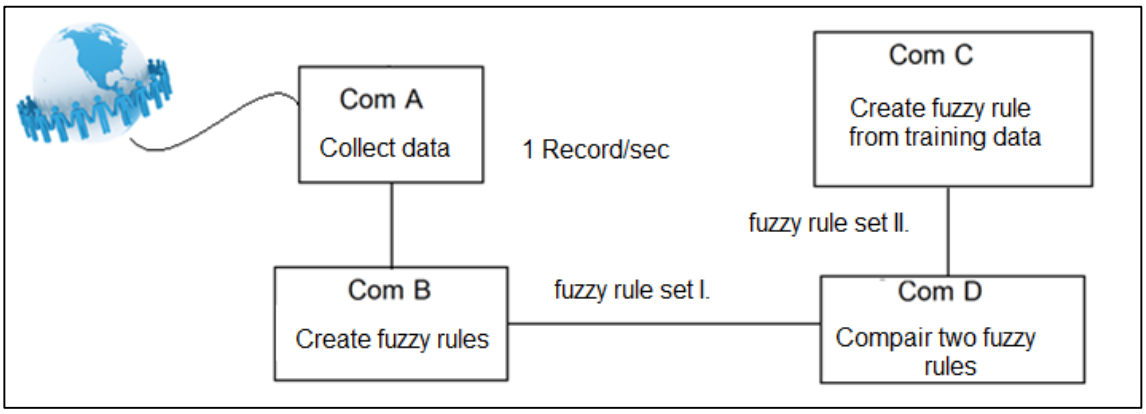



**Figure 2.7** Architecture of NIDS [14]

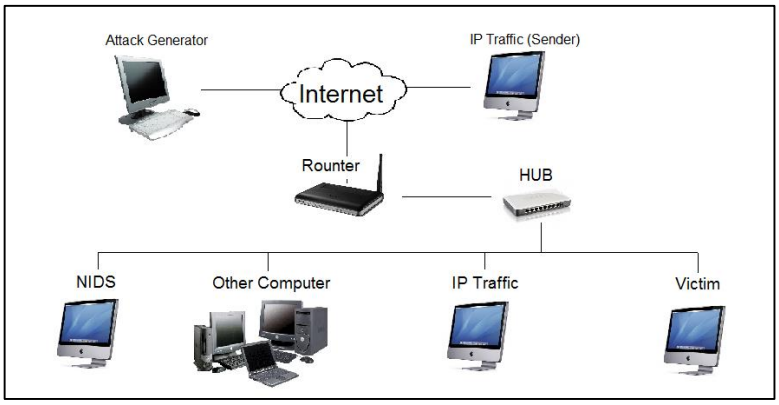



**Figure 2.8** Network topology for simulation [14]

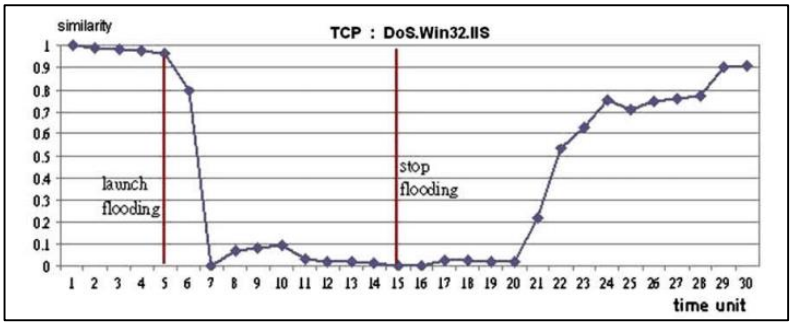



**Figure 2.9** Similarity degradation during flooding for DoS.Win32.IIS [14]

**Table 2.8** Feature list of real-time network IDS for large-scale attacks based on an incremental mining approach [14]

| # | Protocol | Feature |
|---|---|---|
| 1 | TCP | source IP+SYN count |
| 2 | TCP | source IP+URG_Flag+URG_data count |
| 3 | TCP | Source IP+ACK-Flag+ACK count |
| 4 | ARP | Source IP+ARP count |
| 5 | IP | Destination IP slots hit |
| 6 | IP | Header length 1=20 count |
| 7 | IP | MF_Flag count |
| 8 | IP | (total length > 1400\|\|<40)&&TTL=64 count |
| 9 | IP | Checksum_error count |
| 10 | TCP | ACK_Flag+ACK count |
| 11 | TCP | Checksum_error count |
| 12 | UDP | Same_length_interval count |
| 13 | ICMP | Type error count |
| 14 | ICMP | Checksum_error count |
| 15 | ICMP | Checksum_error count |
| 16 | ICMP | Length>1000count |

Komviriyavut et al. [15] proposed a real-time detection. They used a packet sniffer to sniff the packets in the network every 2 seconds and preprocessed it into 13 features by counting the number of connections between two IP addresses every 2 seconds [Table 2.9]. They also used the decision tree algorithm to classify the data. In order to evaluate the performance, they collected the normal data from the network traffic in the Department of CPE from KMUTT. They simulated the attacks in a closed environment by using attack tools which consisted of 18 types of attacks [Table 2.10]. The dataset could be categorized into 3 types; DoS, Probe and normal data. The result showed that this algorithm had 97.5 percent of the detection rate. This technique was efficient to be used in an actual network environment in terms of speed, memory consumption and CPU consumption.

Examples of the record of the normal network data from the preprocessing phase are shown below.

2138,33,33,4,4,644,2136,0,0,0,0,0,0,Normal
12,2,2,0,0,1,12,0,0,0,0,0,0, Normal

**Table 2.9** Features in online dataset [15]

| No. | Feature Description | Data Type |
|---|---|---|
| 1 | Number of TCP packets | Integer |
| 2 | Number of TCP source ports | Integer |
| 3 | Number of TCP destination ports | Integer |
| 4 | Number of TCP fin flags | Integer |
| 5 | Number of TCP syn flags | Integer |
| 6 | Number of TCP reset flags | Integer |
| 7 | Number of TCP push flags | Integer |
| 8 | Number of TCP ack flags | Integer |
| 9 | Number of TCP urgent flags | Integer |
| 10 | Number of UDP packets | Integer |
| 11 | Number of UDP source ports | Integer |
| 12 | Number of UDP destination ports | Integer |
| 13 | Number of ICMP packets | Integer |

Kachurka and Golovko [16] proposed a neural network approach for real-time network intrusion detection. This algorithm could detect the attacks without the training dataset. This experiment considered three different types of the attacks: tcp scan, sysn flood and udp flood (500 records of each attack). The feature names of each record were timestamp, duration of connection in seconds, source's and destination's IP-addresses, name of the service used, port number, the number of bytes transferred and the result flag of the connection. They used both KDD99 dataset and real-time dataset to evaluate the algorithm. This technique was able to detect unknown attacks at least 97% of the detection rate for each type of the attacks (use the KDD99 dataset to evaluate).

Casas et al. [17] proposed Unsupervised Network Intrusion Detection (NIDSs) using Sub-Space Clustering Algorithm and Multiple Evidence Accumulation Algorithm. The NIDSs was able to detect attacks without the training dataset. The system was tested in an offline environment (with the KDD99 dataset) and an online environment. In the online environment, they used the traffic trace from the MAWI repository of the WIDE project and the METROSEC project. These two network traces were generated over the past ten

years. They preprocessed the data network into 9 features [Table 2.11]. The algorithm could be classified into two classes which were a positive class (attack) and a negative class. The result showed that 90% of the attacks were correctly detected.

**Table 2.10** Attack names in the dataset [15]

| No. | Data | Tools (to Generate) | Category |
|---|---|---|---|
| 1 | Smurf | Smurf.c | DoS |
| 2 | UDP Flood | Net Tools 5 | DoS |
| 3 | HTTP Flood | Net Tools 5 | DoS |
| 4 | Jping | Jping.c | DoS |
| 5 | Port Scan | Net Tools 5 | Probe |
| 6 | Advance Port Scan | Net Tools 5 | Probe |
| 7 | Host Scan | Host Scan 1.6 | Probe |
| 8 | Connect | NMapWin 1.3.1 | Probe |
| 9 | SYN Stealth | NmapWin 1.3.1 | Probe |
| 10 | FIN Stealth | NmapWin 1.3.1 | Probe |
| 11 | UDP Scan | NmapWin 1.3.1 | Probe |
| 12 | Null Scan | NmapWin 1.3.1 | Probe |
| 13 | Xmas Tree | NmapWin 1.3.1 | Probe |
| 14 | IP Scan | NmapWin 1.3.1 | Probe |
| 15 | ACK Scan | NmapWin 1.3.1 | Probe |
| 16 | Window Scan | NmapWin 1.3.1 | Probe |
| 17 | RCP Scan | NmapWin 1.3.1 | Probe |
| 18 | Normal | Actual Environment | Normal |

**Table 2.11** Features used in NIDSs [17]

| No. | Feature Description | Abbreviation |
|---|---|---|
| 1 | Number of source IP | nSrcs |
| 2 | Number of destination IP | NDsts |
| 3 | Number of TCP source ports | nSrcPorts |
| 4 | Number of TCP destination ports | nDstPorts |
| 5 | Ratio of number of sources to number of destination | nSrcPorts/nDstPorts |
| 6 | packet rate | nPkts/sec |
| 7 | fraction of ICMP packets | nICMP/nPkts |
| 8 | number of SYN packets | nSYN/nPkts |
| 9 | average packets size | avgPktsSize |

**Table 2.12** Summary of Online IDS

| Year | Author | Algorithm | DR(%) | FP(%) | Number of Features | Output |
|---|---|---|---|---|---|---|
| 2002 [11] | Labib and Vemuri | NSOM | - | | 10 | Normal, DoS |
| 2005 [12] | Amini et al. | Neural Network (ART and SOM) | 97.427 | 1.99 | 27 | Normal, Attack |
| 2007 [13] | Pukkawanna et al. | BLINd classification | 100 (accept host scan) | 0 (accept host scan) | 5 | SYN Flood, ICMP flood, Port scan Host scan, UDP flood and smurf |
| 2009 [14] | Su et al. | Fuzzy association rules | N/A | N/A | 16 | Normal, DoS |
| 2009 [15] | Komviriyavut et al. | Decision Tree and Rule Based | 97.5 | 0.6 | 13 | Normal, DoS, Probe |
| 2011 [16] | Kachurka and Golovko[14] | Neural Network | N/A | N/A | 16 | Normal, Attack |
| 2012 [17] | Casas et al. | Clustering | N/A | N/A | 9 | Normal, Attack |

## 2.2 Background Study

**2.2.1 Artificial Intelligence (AI)** [18]. Major AI researchers and textbooks define the field as "*The study and design of intelligent agents*" where an intelligent agent is a system that learns from giving knowledge and takes action that maximizes its chances to achieve its goal.
John McCarthy : "the science and engineering of making intelligent machines"

**2.2.2 Machine Learning** [19], a branch of artificial intelligence, is about the construction and study of systems that can automatically learn from experiences and get more accurate results. The definition of the machine learning is described as follows:

> Arthur Samuel : "*Field of study that gives computers the ability to learn without being explicitly programmed*"

> Tom M. Mitchell : "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*"

The learning process of the machine learning can be categorized into four types of the machine learning described as follows: [20]

1. Supervised learning: during the learning process, the system will be told by the training dataset what is correct and what is not correct.

2. Unsupervised learning: during the learning process, the correct answers are not provided; the algorithm will identify similarity of the input data and categorize the similar input together instead.

3. Reinforcement learning: during the learning process, the algorithm will be told what is wrong but not be told what is correct. It has to explore and try out different possibilities until it works out how to get the right answer.

4. Evolutionary learning: biological evaluation can be considered as a learning process such as the process that living things adapt their generation to survive in an environment.

There are many ideas proposed to make the algorithm learn. In this work, we are interested in combining fuzzy logic and genetic algorithms together which is a supervised learning approach.

**2.2.3 Fuzzy Logic** can help in decision making or reasoning in an uncertain situation. From Figure 2.10, the fuzzy value is in a range of completely true and completely false but Boolean logic has only true or false.

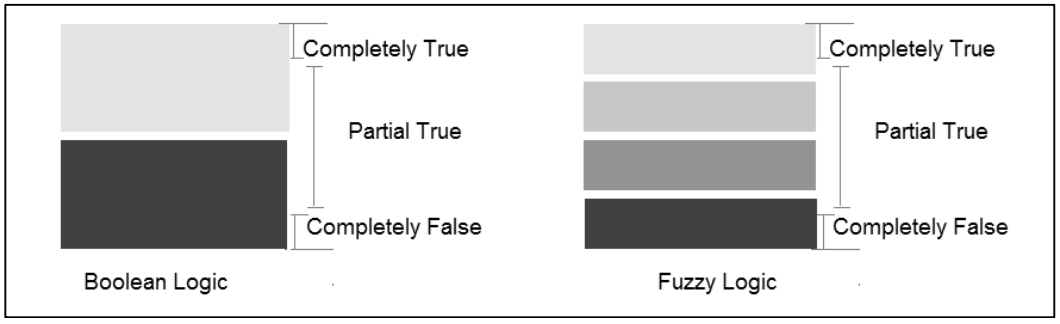



**Figure 2.10** Boolean logic and fuzzy logic

Fuzzy logic uses a membership function to find a solution in an uncertain situation. There are many types of fuzzy functions such as a triangular membership function and a trapezoidal membership function.

For example:

The trapezoidal membership function has three parameters {a, b, c, d} and x is an input value. The fuzzy value (from the input x) will be calculated using the conditions from Figure 2.11.

$$\text{trapezoidal}\ (x;a,b,c,d) = \begin{cases} 0, & x \le a \\ \frac{x-a}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ \frac{d-x}{d-c}, & c \le x \le d \\ 0, & d \le x \end{cases}$$

**Figure 2.11** Trapezoidal membership function [22]

**2.2.4 Fuzzy Rule** contains many fuzzy logics by using an if-then condition. Figure 2.12 presents a fuzzy rule by using many fuzzy logics where $x_i$ is a fuzzy value that is calculated from the fuzzy logic i, $A_i$ is a threshold value from the fuzzy logic i. All input values will be calculated using the fuzzy logic. When all fuzzy values match to rule 1 then the rule will classify it in to Class A.

Rule 1: if $x_1$ is $A_{i1}$ and $x_2$ is $A_{i2}$ and ... then Class A

**Figure 2.12** Fuzzy rule

**2.2.5 Genetic Algorithm (GA)** Genetic algorithms are the evolutionary technique that uses the crossover and mutation operators to solve the optimization problems including NP-hard (non-polynomial) problems. It uses a natural evolution concept of only a "strongest or best solution" will survive among evolution of various populations. The technique does not guarantee an optimal solution. However, it can give a well-enough solution in the given time period. The genetic main algorithm process consists of the following approaches:

- Encoding: each gene is a parameter that a genetic algorithm uses for solving problems. The sequence of the genes is called a chromosome. A chromosome is one solution of that problem.

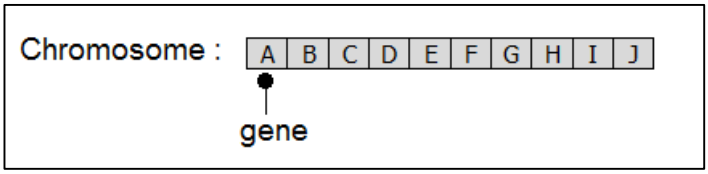
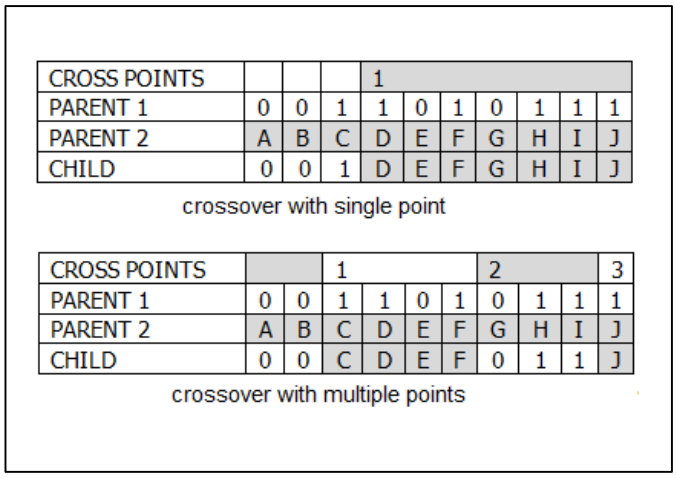



**Figure 2.13** Example of chromosome

- Crossover: the approach to create a new chromosome from an existing chromosome by exchanging parts of the chromosomes (genes) between two chromosomes. In Figure 2.10, parent 1 and parent 2 exchange the chromosomes in a single point and multiple points.



| CROSS POINTS | | | | 1 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| PARENT 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| PARENT 2 | A | B | C | D | E | F | G | H | I | J |
| CHILD | 0 | 0 | 1 | D | E | F | G | H | I | J |

crossover with single point

| CROSS POINTS | | | 1 | | | | 2 | | | 3 |
|---|---|---|---|---|---|---|---|---|---|---|
| PARENT 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| PARENT 2 | A | B | C | D | E | F | G | H | I | J |
| CHILD | 0 | 0 | C | D | E | F | 0 | 1 | 1 | J |

crossover with multiple points

**Figure 2.14** Genetic algorithm crossover multi values

- Mutation: the approach to create a new chromosome from an existing chromosome by randomly choosing the chromosome and randomly changing the gene.
- Evaluation: the function plays an important role in genetic algorithms. It is used to define the value of the chromosome.

### 2.2.6 KDD99 Dataset

KDD99 dataset is a benchmark dataset for an intrusion detection system. It was established in 1999 from MIT Lincoln labs in order to evaluate research results in intrusion detection. The Lincoln labs used the TCP dump to capture the local-area network in the Air Force environment. It was also used with multiple attacks. There were two file versions of the KDD99 dataset: 10% version file (about 500,000 records) and full version file (about 5 million records). Table 2.13 shows a number of the records and a number of the distinct records of each attack type in the 10% version file. Table 2.14 shows 41 features of the dataset.

**Table 2.13** Number of each attack in 10% version file of KDD99 dataset [21]

| Attack | #Original Records | #Distinct Records | Class |
|---|---|---|---|
| normal | 97,277 | 87,831 | Normal |
| back | 2,203 | 994 | DoS |
| land | 21 | 19 | DoS |
| neptune | 107,201 | 51,820 | DoS |
| pod | 264 | 206 | DoS |
| smurf | 280,790 | 641 | DoS |
| teardrop | 979 | 918 | DoS |
| satan | 1,589 | 908 | Probe |
| ipsweep | 1,247 | 651 | Probe |
| nmap | 231 | 158 | Probe |
| portsweep | 1,040 | 416 | Probe |
| guess_passwd | 53 | 53 | R2L |
| ftp_write | 8 | 8 | R2L |
| imap | 12 | 12 | R2L |
| phf | 4 | 4 | R2L |
| multihop | 7 | 7 | R2L |
| warezmaster | 20 | 20 | R2L |
| warezclient | 1,020 | 1,020 | R2L |
| spy | 2 | 2 | R2L |
| buffer_overflow | 30 | 30 | U2R |
| loadmodule | 9 | 9 | U2R |
| perl | 3 | 3 | U2R |
| rootkit | 10 | 10 | U2R |
| **Total** | **494,020** | **145,740** | |

Examples of the data records in the KDD99 dataset:

0,tcp,http,SF,241,261,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,34,169,1.00,0.00,0.03,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,other,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,1,0.14,0.00,0.86,1.00,0.00,1.00,0.00,255,1,0.00,1.00,0.00,0.00,0.13,0.00,0.87,1.00,satan.
0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,510,510,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.
0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,132,8,0.00,0.00,1.00,1.00,0.06,0.07,0.00,255,8,0.03,0.06,0.00,0.00,0.00,0.00,1.00,1.00,neptune.
0,udp,private,SF,28,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,34,34,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,1,0.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,teardrop.

Network attacks fall into four main categories.

- **Denial of Service (DoS)** is a network attack that causes computer resources to be unavailable. DoS can happen from a person or multiple people. The target of the DoS attack is to serve a host on a high-profile web server such as banks, credit card payment gate way. Attackers attempt to force victims to either reset or consume network resources in order to destroy services. There are many methods used for this attack such as SYN flood, Tear drop attack and Peer to per attack.
- **Port Scan (Probe).** Port scanner is a tool designed to probe a server for an open port. Attackers can use this application to monitor behavior of the target and exploit vulnerability of that target.
- **Remote to Local Attack (R2L).** Attackers send packets to a machine and exploit machine's vulnerability to gain the local access as an authenticated user, such as a password guessing attack.
- **User to Root (U2R).** Attackers will start normal access to a user account and exploit vulnerability in order to gain unauthorized access to the root. In common, this kind of the attack can cause the buffer overflow.

**Table 2.14** Forty one features of KDD99 dataset [21]

| # | Feature | Description | Type |
|---|---|---|---|
| 1 | Duration | duration of the connection. | Cont. |
| 2 | protocol type | connection protocol (e.g. tcp, udp) | Disc |
| 3 | Service | destination service (e.g. telnet, ftp) | Disc. |
| 4 | Dlag | status flag of the connection | Disc. |
| 5 | source bytes | bytes sent from source to destination | Cont. |
| 6 | destination bytes | bytes sent from destination to source | Cont. |
| 7 | Land | 1 if connection is from/to the same host/port; 0 otherwise | Disc. |
| 8 | wrong fragment | number of wrong fragments | Cont. |
| 9 | Urgent | number of urgent packets | Cont. |
| 10 | Hot | number of "hot" indicators | Cont. |
| 11 | failed logins | number of failed logins | Cont. |
| 12 | logged in | 1 if successfully logged in; 0 otherwise | Disc. |
| 13 | # compromised | number of "compromised'' conditions | Cont. |
| 14 | root shell | 1 if root shell is obtained; 0 otherwise | Cont. |
| 15 | su attempted | 1 if "su root'' command attempted; 0 otherwise | Cont. |
| 16 | # root | number of "root'' accesses | Cont. |
| 17 | # file creations | number of file creation operations | Cont. |
| 18 | # shells | number of shell prompts | Cont |
| 19 | # access files | number of operations on access control files | Cont. |
| 20 | # outbound cmds | number of outbound commands in an ftp session | Cont. |
| 21 | is hot login | 1 if the login belongs to the "hot'' list; 0 otherwise | Disc. |
| 22 | is guest login | 1 if the login is a "guest'' login; 0 otherwise | Disc. |

**Table 2.14** Forty one features of KDD99 dataset [21] (Continued)

| # | Feature | Description | Type |
|---|---|---|---|
| 23 | Count | number of connections to the same host as the current connection in the past two seconds | Cont. |
| 24 | srv count | number of connections to the same service as the current connection in the past two seconds | Cont. |
| 25 | serror rate | % of connections that have “SYN’’ errors | Cont. |
| 26 | srvserror rate | % of connections that have “SYN’’ errors | Cont. |
| 27 | rerror rate | % of connections that have “REJ’’ error | Cont. |
| 28 | srvrerror rate | % of connections that have “REJ’’ error | Cont. |
| 29 | same srv rate | % of connections to the same service | Cont. |
| 30 | diff srv rate | % of connections to different services | Cont. |
| 31 | srv diff host rate | % of connections to different hosts | Cont. |
| 32 | dst host count | count of connections having the same destination host | Cont. |
| 33 | dst host srv count | count of connections having the same destination host and using the same service | Cont. |
| 34 | dst host same srv rate | % of connections having the same destination host and using the same service | Cont. |
| 35 | dst host diff srv rate | % of different services on the current host | Cont. |
| 36 | dst host same src port rate | % of connections to the current host having the same src port | Cont. |
| 37 | dst host srv diff host rate | % of connections to the same service coming from different hosts | Cont. |
| 38 | dst host serror rate | % of connections to the current host that have an S0 error | Cont. |
| 39 | dst host srvserror rate | % of connections to the current host and specified service that have an S0 error | Cont. |
| 40 | dst host rerror rate | % of connections to the current host that have an RST error | Cont. |
| 41 | dst host srvrerror rate | % of connections to the current host and specified service that have an RST error | Cont. |