CHAPTER FIVE
SPECIFICATION
AND DESIGN
NIDSMAM
CHAPTER 5
SPECIFICATION AND DESIGN NIDSMAM

5. INTRODUCTION

Network-based intrusion detection systems (NIDS) in order to list out attackers, are attempting to expose network and service vulnerabilities, NIDS carry out deeply packet analysis. The main intention of designing hybrid NIDS can help out in recognizing misuse and anomaly patterns and collecting applied data to decision support system to analysis and give ultimate report results. Hybrid NIDS devices may alert users about probable attacks and/or take predefine accessible actions to aid diminished the attack by observing network traffic in real time. To give an extra layer of protection above and beyond access control devices such as a firewall done by NIDSs, this can be a valuable addition to the security arsenal. The one of features of this system that tries to make balance while creating an extra controllable hybrid NIDS deployment without initiating extra risk. This attempt to detect the chance of false negative conditions without raising the number of false positives. However, we needed to analyze how this hybrid network intrusion detection system detect attacks then we can understand the results related with activities by using the analysis results gained from misuse detection system and anomaly detection system by analysis results via DSS.

In this work we design NIDSMAM depended on multi-agents, using JESS, Snort rules, Adaptive threshold algorithm and decision support system DSS. All these factors interact with each other to create a security system is easy and light weight and free from complications. Through the study, of literature survey which was reviewed in the chapters 1, 2, 3, 4, there are enormous and wide efforts from researchers and students to search for the best means to protect the security of the network from intrusions and unauthorized access. In NIDSMAM we design hybrid systems includes two main approaches misuse detection and anomaly detection. Where misuse detection detects known attacks and also anomaly detection detect unknown attacks. This is equation complete achieves by interactions between multi-agents using snort rules, JESS, and adaptive algorithm, and achieves the final report by analysis results via DSS.
5.1 SYSTEM FEATURES AND CAPABILITIES
We recommend the following features and capabilities in NIDSMAM, in defending against network attacks:

1. Integration multi-agents inside the proposed system supplies proactive detection types of attacks.

2. Distributed system where there are no central applications by using the JADE. Jade allows us to create the system which is independent and distributed.

3. The ability to detected false positives and negatives according to security detection strategies that used in the proposed system.

4. The system is capable to detect the number of network activities with high performance rates, and real time detection ability this is depended on scalability and flexibility of all elements the system.

5. Employing security strategy enables depicting and stopping attacks the early stages before distributing across the network.

6. Employing multi-agents for detection known and unknown attacks will contribute to the remove huge amount of data by distributed between misuse detection agents and anomaly detection agents. This is done by building easy and correct glimpses and light weighted system.

7. Using of snort rules to detect the amount of known attacks and malwares patterns by convert snort rules to Jess format.

8. Using of Rete algorithm to matching pattern between facts and snort rules, using this algorithm to faster achieve the task because it takes advantage of structural similarity in rules.

9. NIDSMAM elements have the capable to update the database according to the results of its analysis and information which send by central agent, as well as system administrators can update by the screen monitoring.

10. DSS analysis results for both of misuse detection agents and anomaly detection agents to appear final report for all results. These results to estimate and make the system up-to-date with requests all the agents.
11. The system using the packet sniffer to capture packet from the network in real-time and logging these packets into the knowledge base to enable security experts to characterize network behavior which facilitate detecting potential attacks.

5.2 REQUIREMENTS FOR NIDSMAM DESIGN

We require all the prerequisites mentioned beneath in the completion of NIDSMAM:

*The proposed system operates inside Microsoft Windows environment.*

**Platform:** platform independent

**JAVA:** Programming Language

**SNORT:** Rules

**NET BEANS 6.8:** Java IDE

**JADE Tool:** Create Multi-agents

**JESS:** Java expert system shell

**JPCAP, WinPcap:** Capture Packets from Network

**JDK 1.6:** Java library (JDK)

**Rete Algorithm:** for compare between snort rules and facts

**Adaptive Threshold Algorithm:** to detect deviation from normal behavior

**MYSQL essential 5.1.53:** storing data

**Navicat091 premium:** GUI for database

**Decision Support System (DSS):** provide a final report results

5.3 Architecture of NIDSMAM using Multi-agents

Network Intrusion Detection System NIDS recognition each incoming packets received by the computer system and with the implementation of the signature matching and the statistical algorithm. NIDSMAM monitors the events occurring in the network and analyzes the events for sign of known attacks or unknown attacks; Figure 5.1 illustrates basic NIDSMAM architecture.
The performance of NIDSMAM can be improved by combining anomaly and misuse analysis. This system proposes ensemble multi-agents with adaptive threshold algorithm and Jess, snort rules-based hybrid network intrusion detection model. The proposed model combines anomaly and misuse-based detection analysis. The agents in the proposed model use rules to check for intrusions, and adaptive threshold algorithms to recognize unknown actions. Each agent in the proposed model has a particular categorization technique and gives its faith about any packet event in the network. These agents collaborate to decide the decision about any event has the ability to generalize and to detect unknown attacks. Experimental results send to DSS to generated final report to suggest that the proposed model is efficient and differs from other intrusion detection models. Figure 5.2 illustrates the architecture of improved NIDSMAM.
5.4 THE ROLE OF MISUSE DETECTION SYSTEM

The technique of misuse detection is an attempt to encode knowledge about attacks as well-defined patterns and monitors for the appearance of these patterns. This technique accurately represents knowledge about its prohibited behavior and tries to identify its occurrence. Most intrusion detection systems organized today to make use of misuse detection as analysis method. Misuse detection investigates for attack traces in the recorded audit data using predefined patterns. The matching rules are called signatures. The meaning of signatures is up to now an investigational method based on expert knowledge and experience. The analysis accomplishment and as a result the receipt of intrusion detection systems in general depend essentially on the objectivity of the deployed signatures. The modeled signatures have to be authenticated and corrected to improve their quality. The vital factors for high detection rates in misuse detection are the accurateness and the topicality of the signatures used in the analysis process. Imprecise signatures confine strongly the detection ability and cause false positives or false negatives. Signatures are obtained from an exploit; this is the program that performs the attack. The exploit represents a series of actions that exploit security vulnerabilities in an application, an operating system, or a network. Alternatively, the signatures describes the rules about how traces of these actions can be found in an audit or network data stream.
There are several benefits for misuse detection system: the misuse detection system start protecting network instantly upon installation, decide which intrusive activity the misuse detection system is programmed to alert, the signature definitions are replicated on identified intrusive activity, and the system is simple to understand and the user can rapidly observe the signature database when an alarm fires. The user can link to a particular kind of activity occurring on the network.

In the underneath section, we clarify details about misuse detection system and how can use signatures by convert snort rules into Jess rules and comparing snort rules with facts by using pattern matching through Rete algorithm. Can be explaining all these operations in misuse detection agents in the following.

5.4.1 Concept of Misuse Detection Agents

It is the approach used in improved NIDS for detection of known attacks. This is achieved by matches network packets with the Snort rules database, if the match is found is raised an attack. Misuse detection agents analysis packets arriving from network connection for possible attacks, and trying to match their characteristic with those contained in the rules stored in the rule base.

In this way, the misuse detection only detects the attacks that are already documented. The misuse detection approach consists of three main agents (detection agent, central agent and analysis agent). These agents interaction together to capture packets from the network traffic and determine the types of protocols and rapid response for the analysis of these packets to reduce the response time and speed up detection for improved the accurate performance for the system. We illustrated how to work the misuse detection agents and interact multi-agents with each other to detect the attack and to notification alarm to system administration and to provide the DSS the results of the detection to give a final report to evaluate the system performance. Here, the use detection agent described work to ease configurability and extensibility. Finally, misuse agents developed here is one of the different types of agents included in the multi-agent system. Figure 5.3 illustrates the misuse intrusion detection agent’s architecture.

Misuse detection agents contain:

- Detection agent
- Central agent
- Analysis agent includes:
  - Conversion agent
  - Response agent
  - Alarm agent
  - User interface agent
  - Registration agent

*Figure 5.3 [Illustrates the Misuse Intrusion DetectionAgents Architecture]*

### 5.4.1.1 Detection Agent

Detection agent is work to detect attacks; it is part from misuse detection agents. This agent captures packets from the network using a packet sniffer Jpcap, WinPcap, which scans the packet and creates the knowledge base of the information it contains retrieve one by one packets from knowledge base after stored it as a facts and retrieve these facts from knowledge base and convert into a Jess format as an input to Jess rule engine. In
detection agent, we are using inference rule engine to detect attacks, on the other hand, Jess rule engine needs the standard signatures to detect attacks. Snort organization provides the standard signatures for detect attacks. So, with the help of snort we able to detect the correct attack by convert snort rules into Jess rules using Jess rules engine. Snort rules give as an input after convert to Jess rule to inference rule engine with facts for detect attack, because inference rule engine use Rete algorithm to pattern matching. We have several rules, and also number of packets received per second. So, each packet should be compare with each snort rule and its very time consuming. Rete algorithms help us to reduce the required time for pattern matching. Also, the result of each packet is given as an input to the central agent and packet information. Figure 5.4 illustrates detection agent architecture, this agent consists of:

**Detection Agent**

![Detection Agent Architecture](image)

**Figure 5.4 [Illustrates Detection Agent Architecture]**

1. **Packet Sniffer:** a computer network, often simply referred to as a network, is a collection of hardware components and computers interconnected by communication channels that allow sharing of resources and information. For sharing the resources from different network, a security is the main concern. In the field, of networking the area of network security consists of the provisions and policies adopted by the network
administrator to prevent and monitor unauthorized access, misuse, modification, or denial of the computer network and network-accessible resources. There are various kinds of attack that can be affecting the system. So, we developed a system which detects this kind of attack.

A packet sniffer is a program that allows eavesdropping on traffic traveling between networked computers. The packet sniffer will capture data that is addressed to other machines and generate a log for received packets into the network. A packet sniffer can be used legitimately by a network or system administrator to monitor and troubleshoot. In its simple form a packet sniffer simply captures all of the packets of data that pass through a given network interface. For that, we used JPcap and WinPcap for capture packets that pass through a given network interface. We used Java as a platform to execute JPcap.dll file. After installation of JPcap and Winpcap that JPcap.dll file is placed at required path like root folder of JDK 1.6 and system32 folder. To implemented packet sniffer we need to use JPcap.jar as API. In packet sniffer, there is a method i.e. receive packet (Packet p). If any transaction is occurred into a network then this method is called using thread. When packet is capture it is handled by thread, one thread for single packet. Using JPcap we can get the list of network interfaces. Network interfaces are nothing but the point interconnection between computer and private and public network, to start the system we have to select proper interface. JPcap is a Java based library that uses the C library libPcap (WinPcap in Windows) which is used to capture packets. It also supports sending raw packets. It is still an effective tool for writing security and network tools.

WinPcap is an open source library for packet capture and network analysis for the Win32 platforms. The purpose of WinPcap is to give this kind of access to Win32 applications; it provides facilities to: capture raw packets, both the ones intended to the machine where it is running and the ones exchanged for other hosts (on shared media), filter the packets according to user-specified rules before sending them to the application, transmit raw packets to the network and gather statistical information on the network traffic. JPcap is a Java library for capturing and sending network packets. Using JPcap, we can develop applications to capture packets from a network interface and visualize /analyze
them in Java. We can also develop Java applications to send arbitrary packets through a network interface.

\[ 
\text{Devices} = \text{JpcapCaptor.getDeviceList}(); 
\]

When we want to capture packets from a network, the first thing we have to do is to obtain the list of network interfaces on the machine. To do so, JPcap provides \text{JpcapCaptor.getDeviceList()} method. It returns an array of network interface list. A network interface list contains some information about the corresponding network interface, such as its name, description, IP and MAC addresses, and data link name and description. Once we obtain the list of network interfaces and choose which network interface to capture packets from, we can open the interface by using \text{JpcapCaptor.openDevice()} method. The following piece of code illustrates how to open a network interface.

\[ 
jpcap = \text{JpcapCaptor.openDevice}(\text{devices[device]}, 2000, \text{false}, 20); 
\]

\text{JpcapCaptor.openDevice()} returns an instance of \text{JpcapCaptor}. We can then call several methods of the \text{JpcapCaptor} class to actually capture packets from the network interface.

\[ 
jpcap.loopPacket(-1, \text{new Sniffer(ljep, type)}); 
\]

Methods to start capturing using the callback method. When calling \text{loopPacket()} method we can also specify the number of packets to capture before the method returns. We can specify -1 to continue capturing packets infinitely. In NIDSMAM to detect the attack by capture the packets coming from the network by using a packet sniffer. When we click on the button start the thread which is responsible of starting the packet sniffer get executed, and packet sniffer starts capturing the packets. The java code for it is as below:

\[ 
\text{Tmpthrd} = \text{new snifferthread(type, getModel(), true);} 
\]
In the above code `Tmpthrd` is a thread and with new `snifferthread` object is created and the thread is start. On that thread we are running the packet sniffer. Figure 5.5 illustrates packet sniffer Java code.

![Figure 5.5 Illustrates Packet Sniffer Java Code](image)

The above figure shows java code of packet sniffer and in that we have a receive method is responsible for receiving the packet from the network.

```java
@Public void receivePacket (Packet packet) {
    System.out.println ("Other packets: "+packet);
}
```

Here, we have a `ReceivePacket` method. In this method, we are capturing the packet, and we are analyzing the packet. Firstly we are partitioning the packet and getting the type of protocols. Figure 5.6 illustrates output packet sniffer.

```
1333342180:171081 /192.168.2.1->/239.255.255.250 protocol (17) priority (0) hop (4) offset (0) ident (46912) UDP 1900 > 1900
```
2. Knowledge Base: a knowledge base provides a means for information to be collected, organized, shared and utilized. They contain a set of data, often in the form of rules that describe the knowledge in a logically consistent manner. In NIDSMAM packet sniffer sniffs the packet from the network. These packets are dynamically converted to rule format and stored on one file. The knowledge base is for maintaining the packet information in the required format for the system.

Following code snippet, will explains the how knowledge base works. The `storePacket(HashMap<String, Object> pack_data)` is responsible for storing the packets in to file called `SnifferDB.txt`. If we see the code of packets are formatted according to protocols and format which is taken place at `storePacket()` method.

```java
static File pack_file = new File("C:/Documents/NIDS new Backup/Sniffer DB/SnifferDb.txt");
public static void storePacket(HashMap<String, Object> pack_data) throws IOException {
    output = new BufferedWriter(new FileWriter(pack_file, true));
    String packet_data = "";
```
packet_data = pack_data.get("src_ipaddr") + " ||| " + pack_data.get("src_port") + "|||" + pack_data.get("direction") + "|||" + pack_data.get("dest_ipaddr") + "|||" + pack_data.get("dst_port") + "|||" + pack_data.get("protocol") + "|||" + pack_data.get("ID") + "|||" + pack_data.get("seq") + "|||" + pack_data.get("content") + "|||" + pack_data.get("size") + "|||" + pack_data.get("SYN") + "|||" + pack_data.get("RST") + "|||" + pack_data.get("startTime") + "|||" + pack_data.get("Date");

output.append("r\n");
output.append(packet_data);

This knowledge base is collected and organized accordingly with the system. Knowledge base is shared among the whole system and also used for creating the facts for misuse detection module.

3. Generating of Facts: The Knowledge base stored the incoming packet from packet sniffer. These packets are dynamically converted to rule format. That is the Jess rule format. Those converted rule formats are stored on one file and called as facts. These facts are to read dynamically from Jess rule engine. The misuse system uses these facts in the Jess rule engine, with these facts misuse detects attack type in the system. Following code snippet explains the facts format and conversion of packets to facts. Code having the `pack_data.get()` method for getting the packets from packet sniffer. These packet data are formatted to Jess rule engine data. So, that it would becomes the facts. The `executeCommand()` method for executing and storing facts to one file which is used for detecting the attacks in misuse agents.

```
engine.executeCommand("(deffacts packet (protocol " + q + pack_data.get("protocol") + q + ")" + 
" (source_ipaddr " + q + pack_data.get("src_ipaddr") + q + ")" + " (source_port " + q + pack_data.get("src_port") + q + ")" + 
```
"(direction " + q + pack_data.get("direction") + q + ")
(destination_ipaddr " + q + pack_data.get("dest_ipaddr") + q + ")" + " 
"(destination_port " + q + pack_data.get("dst_port") + q + ")" + "(
(content " + q + pack_data.get("content")+ q + ")" + 
" (ID msg:" + q + pack_data.get("ID") + q + "))");

4. Snort Rules: the data packets that are relocated through the network are the cause of worry for the security threats, dangers to the security system, based on the principle of trace the particular configuration or pattern. All these patterns are stored in the rules. These rules are written and tested by the Vulnerability Research Team of Snort organization. Which make possible the supply system with the essential information in case there were tries to breakthrough to take the needed measures. So, the first step in our project is to convert the snort rules in the Jess rules. This is example of snort rules:

Alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ATTACK-RESPONSES directory listing"; flow: established; content:"Volume Serial Number"; classtype: bad-unknown; Sid: 1292; rev: 9 ;)

The following Figure 5.7 shows the snort rules format before converting in to Jess format.
5. Snort Rules Convert to Jess Rules: in NIDSMAM we are using snort rules to contribute to the detection of known attacks in misuse detection agents. There are several advantages where the implemented snort rules in Jess rules are modulation into a Java code which would raise the Jess rule engine. When the Jess code is modulation in Java all the several advantages of the Java software will be present for creating the system robust, platform independent. Figure 5.8 illustrates Snort rules to Jess format.

![Figure 5.8](image)

Snort providing signatures into a number of files to collect all rules files into .clp file. In .clp file snort rules are converted into Jess format. In misuse detection using Rete algorithm for inference rule engine. In Rete algorithm to compare received packets (facts) with snort rules using Jess format. To get alert after detection, the alert message put into the file. If pattern matching done in LHS the RHS part will execute. In that part we open the alert file, write alert into that file and close it. This means that using the instance of the Rete, rules file (.clp) will be loaded from local memory using execute command function which is present in API. Path of the rules fired can as well as be extracted. On the other hand, instance of Rete, facts (.dat) will be loaded from local memory. The run function has to be called on the instance of Rete to fire the rules. For an example:
File dir = new File("C:/NIDS File Dir/Snort/rules");

FileWriter infstream = new FileWriter("C:/NIDSMAM/Dir/rules/all_rules.clp", true);

BufferedWriter out = new BufferedWriter(infstream);

try{
    if (dir.isDirectory()) {
        File[] rulesfiles = dir.listFiles(new FilenameFilter() {
            @Override
            public boolean accept(File dir, String name) {
                return name.toLowerCase().endsWith(".rules");
            }
        });

        int nfiles = rulesfiles.length;

        In this code, we read the snort rules file. This file is open in the read mode where we read the file rule by rule and then the rule is broken into parts so we can match each part with the facts. We have used the string functions to perform the various actions on the rules we have read from the file. String Tokenizer is used split the rule string and facts are also split in a particular format so we can use Rete algorithm for pattern matching. We have stored these spited rules in the array list. We have iterated over the array list for every single rule in the file. Each rule is in the array list is converted in the Jess format which are also stored in the collections array list. Below Figure 5.9 illustrates Snort rules in Jess format.

        (deffunction Rule1 "attack-responses. Rules" (protocol "tcp")
            (source_ipaddr "$HOME_NET")
            (source_port "any")
            (Direction "->") (destination_ipaddr "$EXTERNAL_NET")
            (destination_port "any") (Content "")
            (ID )=> (printout t alert, "Volume Serial Number" crlf))

        }
    }
}
6. JESS Rule Engine: Jess is an acronym for Java Expert System Shell. The selection of software to be utilized to illustrate the snort rules plays a vital role in the measures taken for the inclusive system itself. This kind of software affords support to the system. An erroneous option makes a superior system takes the phase of incomplete in the performance of his responsibilities.

Jess is a Java based rule matching engine. Jess software is used for the implementation of the Snort rules to provide a system that is capable of observing the intrusions on a network. Jess which is fundamentally a rule based engine invented on the Java platform has all the vital distinctiveness of Java. Java is called a platform independent language because system specific libraries that provide the required programming language support for any software to build up a complete system do not exist in Java. Alternatively, it reduces the overhead of the addition of necessitated libraries for the advanced system to execute or to recompile the software every time there is a modification in the system that utilizes the software. Entire this distinctiveness that approach with Java as well as existing even in Jess since of which the system becomes a platform independent system.

The basic step in NIDSMAM to detect misuse attacks we have to compare received packets with a database of snort rules. If we used simple pattern matching concept it takes too long time for several rules. So, the use of Jess rule engine to reduce the time and to
increase productivity. We can use Jess scripting language as well as we can embed Jess into Java code for that Jess provides API. In inference rule engine, the Rete algorithm is used to pattern matching. In Jess.jar there is Rete class. In this Rete class, Rete algorithm is implemented as shown that in code below. We just create an instance of Rete class. Rete class provide execute commend () method, using this method we can execute various commends from Jess rule engine using the following statement we can upload the rules file into inference rule engine.

```java
Engine.execute commend("(batch"c:/documents/rules/all-rules.clp")");
```

This file is always uploaded for each packet. To upload incoming packet into Jess rule engine, we have convert packet into Jess fact dynamically. We can upload fact using second statement of execute commend () method of Rete class. After loading rules and facts into inference rule engine following statement help to execute pattern matching.

```java
engine.run ();
```

This statement returns the integer value “0” or “1”. If pattern matching is done then its return “1” or its return always “0”.

**7. Inference Rule Engine:** inference rule engine is a part detection agent which performed a pattern matching. Jess is a rule based program or software that employs a certain if – else statements on a set of provided data. Rule matching algorithms are typically passive in terms of the memory usage as of the time taken by them to relate the rules and these flows can be eliminated by the Rete algorithm. Rete algorithm in Jess can be utilized on the foundation of nodes. To accomplish this by developing a network of nodes meaning one or more tests those are noticed on a rule in the Left Hand Side (LHS). The facts or data from the system being tested are removed from the working memory and then sent across the network for processing. The nodes offered at the end of the network are the rules singly written in Jess. Rules are said to be activated if they pass all the required tests that are existing on the LHS of a unique rule. The activated rule may be related with an RHS which is performed or fired. Rete algorithm makes Jess one of the better rules engines subsisting today. Inference rule engine takes two kinds of an input for pattern matching. It is the central part of a rule engine and controls the whole process of applying rules to the working memory to obtain the outputs of the system. After
loading rules and facts into inference rule engine following statement help to execute pattern matching.

\[ \text{engine.run();} \]

This statement returns the integer value “0” or “1”. If pattern matching is done then its return “1” for true or its return always “0” for false. As per the result inference rule engine, forwards this result to the central agent which do further remaining work.

\[ \text{Rete engine = new Rete();} \]
\[ \text{result= engine.run();} \]

\text{engine.run() method is inbuilt in Rete class. We call that method for pattern matching.}

The purpose of the use pattern matching between facts and the stored rules to provide a high-level performance to given other agents, and enhances coupling between the other agents and the system.

5.4.1.2 Central Agent

Central agent is the module which decided on whether the incoming packets should go to analysis agent or to be sent to the anomaly agents in accordance with the result from detection agent. This module also serves as an intermediate between detection agent and analysis agent. In NIDSMAM central agent plays the important role to maintain flow of received packets. Central agent received message from detection agent, in that message results of inference rule engine and packet information it is taken a decision whether the received packets is gone through analysis agent or anomaly agents. On the basis of results of inference rule engine if the received packets are detected by detection agent then central agent pass the packet information to analysis agent, i.e. it is of known attack. If the packet is not detected by detection agent then central agent pass the information of the packet to anomaly detection agents.

5.4.1.3 Analysis Agent

In NIDSMAM analysis agent is the main component of misuse detection agents, is working for to misuse detected packets. Analysis agent is not a single component, but it includes various agents. It is the combination of different type of agents, and it perform the combine tasks. When the central agent sends the attack information to the analysis agent, this agent takes various actions on that attack. Analysis agent is responsible to
show attack information to the user and make alarm on the system. This information then stored in the database. Analysis agent contains the following agents:

- Conversion agent
- User interface agent
- Alarm agent
- Response agent
- Registration agent

Below more details about all agents for analysis agent: Figure 5.10 illustrates the analysis agent architecture.

**Analysis Agent**

1. **Conversion Agent**: central agent gives the input to the conversion agent. Conversion agent is an act as a communication bridge between central agent and analysis agent, and providing them information required. It receives packets information from detection agent with the help of central agent. Central agent takes the decision to send packets to conversion agent. It is pass the information of packet from central agent to response agent, and to user interface agent to perform some action. This packet information used for making analysis and store into the database inside registration agent. The following
code will show how conversion agent receives the packets information and what conversion agent does.

```java
public AID Response;
getAID();
```

Each agent is identified by an “agent identifier” represented as an instance of the `jade.core.AID` class. The `getAID()` method of the agent class allows retrieving the agent identifier.

```java
hm_packet.put("time_stamp", obj.getTime_stamp());
hm_packet.put("pakc_info", obj.getPack_info());
hm_packet.put("date stamp", obj.getDate_stamp());
InputForResponseAgent.setMDTableInfo(hm_packet);
```

The packet information is stored into the `Hashmap object hm_packet`. And this `hashmap (hm_packet)` to the response agent. Conversion agent forwards this message to response agent. This message contains time stamp of when that packet is detected. The information about the packets and date stamp. Now conversion agent put this information into a hash map and sends this information to response agent. The following code will show how it transfer the message to response agent

```java
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setLanguage(codec.getName());
msg.setOntology(ontology.getName());
getContentManager().fillContent(msg, new Action(Response, action));
msg.addReceiver(Response);
this.send(msg);
```

`SendMessageResponse()` method is used to send the information. `GetContentManager() .fillContent` method is used to get the message with a response action. `this.send (msg)` method is used to send the message (packets) to response agent. In the above code, we have sent message (packets information) to response agent. So the conversion agent is receiving packets and sending it to the response agent. Conversion agent also sends the packets information to the UI agent.
2. Response Agent: it is part of analysis agent used to give a response on attack detected. It receives packets information from conversion agents and also provides notifications to UI agent and sends the informing to alarm agent to give out corresponding alarms. The code to receive information from conversion agent is shown as below.

Public ReceiveMessages (Agent a)

Response agent sends the received information to UI agent. The code for PING to UI agent is shown below.

```java
void sendPingtoUIAgent (AgentAction action) {
    ACLMessage msg = new ACLMessage (ACLMessage.REQUEST);
    msg.setLanguage (codec.getName());
    msg.setOntology (ontology.getName ());
    GetContentManager ().fillContent(msg, new Action(Ui, action));
    msg.addReceiver (Ui);
    this. Send (msg);  
}
```

In the above code `void sendPingtoUIAgent(AgentAction action)` is responsible for invoking the `getContentManager().fillContent(msg, new Action(Ui, action));` this method just forwards the message to the UI agent. The code for PING to alarm agent is as follows:-

```java
void sendPingtoAlarmAgent() {
    ACLMessage msg = new ACLMessage (ACLMessage.REQUEST);
    msg.setLanguage (codec.getName());
    msg.setOntology (ontology.getName ());
    msg.addReceiver(Alarm);
    msg.setContent("Play Beep");
    this.send(msg);
    //System.out.println("Contacting alarm... Please wait!");  
}
```

A function `SendPingToAlarmAgent()` is called by response agent. `Send(msg)` method send a message to the alarm agent.
3. **Alarm Agent**: it is a part of analysis agent. This agent sends notification to user/system administrator if any attack found to perform some action. After getting the message by response agent, alarm message execute the alarm tune or sound beep as soon as it receives detected packets. Alarm agent sends this information to the registration agent to store into database. When it receives a detected packet it executes a following code:

```
public static void play_beep () {
    Toolkit.getDefaultToolkit ().beep ();
}
```

Toolkit is an abstract window toolkit class. `Toolkit.getDefaultToolkit ().beep ();` this function beep the alarm. The beep sound comes when the attack is detected by the system. Beep sound comes to the background of the system. Alarm agent also performs some other tasks, it forwards all received packets to next registration agent and UI agent.

4. **User Interface Agent**: this agent interacts between the users with the system (e.g., System administrators), the aim of integrating them flexibility into the multi-agents architecture. UI agent receive the attack information from response agent and alarm agent, conversion agent and depending upon the information received from these three types of agents UI agent take action. As soon as user interface agent received the message from alarm agent, user interface agent display the alert message dialog box which indicates the attack is detected. On this dialog box some options are provided to the user when any attack is detected, like shutdown internet connection, machine shutdown, shutdown NIDS or ignore. A Figure 5.11 shows the misuse alarm dialog box. If the system administrator selects any one of them, then that particular action is taken by UI agent. User interface agent also gets the information packet from response agent; this packet information is used to show current detected packets onto table. If the system administrator wants to see the historical information of detected packet which is stored in the database. On the other hand, when alert message generated then user has to take action on attack, this detected packet information is store into a database for displaying results on DSS.
As the above diagram shows the action user can take when the attack is detected by the system. UI agent performs the task according to the user selection.

5. Registration Agent: to display the results on DSS and other misuse table we need to maintain all data of detected packets. So, registration agent is work to store information about detected packets into databases which received from the alarm agent and UI agent. That information is use to show the analysis part on DSS and to show the simple packet information in table. This agent logs all the intrusion information includes:

- Packet information
- Type of attack
- Received Date
- Received Time
- Detect time
- Detect date

This format is stored in the database, and to send DSS to provide a final report results to all the system activities. The code for storing the packet information into a database is as follows:

```java
DB.storeMDPacket(obj.getTime_stamp(), obj.getJess_res(),obj.getPack_info(),
obj.getDate_stamp());
```

5.5 THE ROLE OF ANOMALY DETECTION SYSTEM

Normally network anomalies indicate to the conditions when network operations deviate from the normal behavior. Anomaly detection in an IP network is a very hard work,
because it is dependent upon the type of the data that is available for the analysis. This data can be separated into two basic cases, first is network based, and second is end-user-based. Network based data could be the data which is used by different network devices such as a routers who exchange data to maintain routing tables etc. End user based data points to the transmission control protocol TCP, user datagram protocol UDP related data that include information that is specific to the end application. There are multiple reasons for arising network anomalies like Denial of service DoS attacks, network overload and network intrusions that interrupt the normal delivery of network services.

DoS attacks are essentially interference that reduces or removes the availability of service. A service is any aspect of a computer system's functioning that provides benefit to a user. Service is denied if a computer crashes or its bandwidth is used up or its hard disk is filled. The commonest types of DoS attacks include connecting server, CPU cycles or resources, disabling web traffic, and mail bombs. In case of network intrusions the malicious entity could hijack network bandwidth by flooding the network with unnecessary traffic, hence thirsting other legitimate users; this anomaly would result in heavy traffic volumes. There are several benefits for anomaly detection system: anomaly based IDS can monitor the network and trigger an alarm if an event outside known normal behavior is detected, the system is based on dedicated profiles, it is very hard for an attacker to know with sure what activity he can do without setting off an alarm. Nevertheless, the intrusive activity is not based on particular traffic that represents known intrusive activity as in a knowledgebase NIDS, key element of intrusion detection in which deviations from normal behavior indicate the presence of intentionally or unintentionally excited attacks, anomaly-based detection algorithms that enable detecting novel attacks and malwares and anomaly detection results by adapting the detection thresholds of any anomaly detection system in accordance with the observed input characteristics. In this section, we explain details for anomaly detection system that is including:

5.5.1 Concept of Anomaly Detection Agents

Anomaly detection agents are one of the important multi-agents in NIDSMAM which observes and learns the network deviations and network traffic. Samples of sequences of
normal events in the system are collected, and make a set of observations. Those observations are recognized unusual and potentially dangerous events in the network. Anomaly detection agents are a module involves the multi-agents which observe the system for deviations from normal behavior. The detection of abnormal behavior of the system indicates the presence of intentionally or unintentionally excited attack. In order to determine what attack traffic is, the system must be recognize normal system activity. If any deviation from the set of normal patterns then it is considered as anomalous event. This set of observations we can calculate or we can give them manually for setting observations rules. The anomaly detection agents include the interaction of agents with each other with the implement of the adaptive threshold algorithm to detect whether there was a deviation from the normal level. Previously, was a reference in the misuse detection agents to inference rule engine does a pattern matching between incoming facts and snorts rule using Rete algorithm and send results to the central agent. Central agent forwards that packet according to results send by the inference rule engine. In pattern matching is done successfully it return true and send packets to the analysis agent for detection of known attack and if pattern matching false it forwards packets for anomaly agents to detect unknown attack by using adaptive threshold algorithm. Anomaly agents contain at different types of agents to perform various tasks:

- Information Agent
- Sensor Agent
- Statistic Agent
- Decision Agent

These agents interaction together to capture packets from the network traffic and determine the types of protocols and rapid response for the analysis of these packets to reduce the response time and speed up detection, if there was a deviation is abnormal attacks and remove the complexities from the system. On the other hand, accurate in the performance and detected the false alarms rate. Now explain roles these agents in details, Figure 5.12 illustrates anomaly detection agents architecture.
The above figure shows anomaly detection agents with multi-agents architecture. Anomaly agents detect attack in statistic agent using other agents like information agent and sensor agent. This can be accomplished in several ways, but NIDSMA approach is using adaptive threshold algorithm with multi-agents.

1. **Information Agent:** information agent is used for the getting the data and processed information in the system from central agent. This agent is one kind of interface which keeps track of information required for anomaly detection. It is acting as the unit data repository for anomaly agents. After the detection agent capture packets and completed all procedures that have been described in section misuse detection agents; the detection agent send the packet information and all results for pattern matching to central agent. At this time the central agent checking the pattern matching results if the result is “0” then we send the packet information to the information agent which is part of anomaly agents. And if it is “1” then we send information to analysis agent which is part of misuse agents. In NIDSMSM the information agent receives the packet information from central agent and collects all the packets for a particular time interval “10sec”. Means we put a fixed time period for the information agent that is “10sec” so that it collects all packets from central agent in that time period and forward these packets to the sensor agent which has
a main role of sorting out the packets according to their protocols. The following code shows information agent collects packets in given time of period.

```java
addBehaviour (new TickerBehaviour(this, 10000) {
    protected void onTick() {
        InputForAnomaly obj = new InputForAnomaly();
        obj.setList(list);
        sendMessageSensor(obj);
    }
}
```

addBehaviour (new TickerBehaviour(this, 10000), This method collects incoming packets information in “10sec”. In this agent, following code snippet explains, there are two methods which actually work for this agent. First method is that, public void ReceiveMessages(Agent a), which receives information from central agent in the message format. If we see the code neatly, then we find that method extractContent(msg) which extracts the message information and accordingly, adds this information to list for storing. Second method void sendMessageSensor(AgentAction action) which sends the information to sensor agent. This method contains the msg.addReceiver(Sensor); which adds the sensor agent with massage. After adding the sensor agent fillContent(msg, new Action(Sensor, action)); executes which means that information in terms of message would be filled and finally this.send(msg); runs, this is message information is send to sensor agent. Also, we have given the time interval like “10sec”. With this time interval we collect the bunch of packets which will be input for adaptive threshold algorithm. This bunch of packets will send to the sensor agent. So, that in sensor agent it would separate the packets accordingly with protocol.

```java
public ReceiveMessages(Agent a) {
    ACLMessage msg = myAgent.receive();
    if (o.flag1 == 0) {
        o.flag1 = 1;
        list.clear();
    }
```

[177]
The above code shows that information agent received packets information from central agent and sends packet information to the sensor agent.

2. **Sensor Agent:** this agent sense packet from the information agent and pre-process by reducing irrelevant data and remove the unnecessary packet information and extract the independent features. When the information agent send list of packets of sometime interval to sensor agent. The sensor agent received list of packets and checking each packet protocols and according to each packet the sensor agent segregate the protocols (TCP SYN, TCP RST, TCP REST, UDP, and ICMP), we collect information for each protocol. In the collection of packets if we having packets of UDP protocol then we will add that packets into UDP packet list, if it is TCP protocol then we checking if it is having SYN flag “on” or “off”. If it is “on” then we add into TCP SYN packet list, if it is “off” then we checking for RST flag “on” or “off”, if it is “on” then we add into TCP RST packet list. And if it is “off” then we add in to TCP REST packet list. If we having packets of ICMP protocol then we add that packets into ICMP packet list. After distributing this all packets according to the protocols it will send to statistic agent.

If we see the following code, we come to know that; in every if () statement checks and segregates the packet accordance with the protocol. After separation of these 5 protocols, the control of this program goes towards statistic agent for further processing by calling
InputForStatistic o = new InputForStatistic(); and o.setUdp(udp) (for other protocols also); The example given below is a sample for UDP packets. It first checks the packets protocol if it matches it does the further processing. If this condition is true (Udp_Flag == true) it takes UDP_time and Udp_date. The syntax for UDP protocol is shown below:

```java
if ("UDP".equals(obj.getList().get(i).get("protocol"))) {
    if (Udp_Flag == true) {
        Udp_Time = obj.getList().get(i).get("startTime").toString();
        Udp_Date = obj.getList().get(i).get("Date").toString();
        Udp_Flag = false;    }
    udp.add(obj.getList().get(i));    }
```

After done with sorting sensor agent forwards these packets to the statistic agent which performs further task. The code for forwarding this message to the statistic agent is as follows.

```java
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.setLanguage(codec.getName());
    msg.setOntology(ontology.getName());
    getContentManager().fillContent(msg, new Action(Statistic, action));
    msg.addReceiver(Statistic);
    this.send(msg);
```

getContentTypeManager().fillContent(msg,new Action(Statistic, action)); above line of code for filling the content of the message. And msg.addReceiver(Statistic); this.send(msg); this line code sends the packet information to the statistic agent.

3. Statistic Agent: it is responsible for making complex calculations and analysis over the packets data. These packets data are provided from the sensor agent to this statistic agent. According to the each type packets the statistic agent generates the result by using adaptive threshold algorithm. These results about the security status are forwarded to the decision agent to take the decision over detection of packets in anomaly agents. When sensor agent distributing packets into types of protocols and each protocols packets list
will be send to statistic agent. Now, statistic agent collected all this information from
sensor agent and according to types of the protocol it will be passing to appropriate
algorithm. In that time interval if UDP packet found that is meaning UDP packet list size
is greater than “0” then we will pass that UDP packet list to the UDP adaptive threshold
algorithm. If TCP SYN packet found that is means TCP SYN packet list size is greater
than “0” then we will pass that TCP SYN packet list to the TCP SYN adaptive threshold
algorithm. Same procedures for TCP RST, TCP REST and ICMP packets list when we
pass list of packets to adaptive threshold algorithm (ATA) then in (ATA) us calculating
the number of packets. According to adaptive threshold algorithm a threshold is set
depending upon experience. Also, initialized one variable as a counter (i.e. k) if the
number of packets exceeds thresholds that time we increase counter by 1 and when it
consecutively repeated for $k$ times then statistic agent raise an anomaly attack. Following
code snippet explains only for UDP packet:

The method \texttt{void ATA_UDP(ArrayList\<HashMap\<String, Object\\>\> udp)} invokes and
calculate the UDP number of packets, $udp\_threshold$, $UDP\_mean$ and alpha is
calculated using this $(udp\_no\_of\_packets - udp\_threshold) / udp\_threshold$.

If “$(udp\_no\_of\_packets \geq (alpha + 1) * (UDP\_mean))$” satisfies that it is UDP
flooding attack. When this attack is detected the statistic agent sends the packet
information along with time, date and type of attack towards decision agent using
\texttt{sendMessageDecision(obj)};

\begin{verbatim}
void ATA_UDP(ArrayList<HashMap<String, Object>> udp) {
    int udp_no_of_packets=udp.size();
    int udp_threshold = o.getUdp_t();
    int udp_k = o.getUdp_k();
    float beta;
    float alpha;
    int temp = UDP_count;
    if (UDP_count == 0) {
        UDP_count++;
    }
    ...
}
\end{verbatim}
UDP_mean = udp_no_of_packets;

} else {

if (udp_no_of_packets > udp_threshold) {
    alpha = (udp_no_of_packets - udp_threshold) / udp_threshold;
    System.out.println((alpha + 1) * (UDP_mean));
    if (udp_no_of_packets >= (alpha + 1) * (UDP_mean)) {
        UDP_k++;
        System.out.println(UDP_k);
        if (UDP_k == udp_k) {
            Calendar cal=Calendar.getInstance();
            Det_Date=cal.get(Calendar.DATE)+"/"+
                    ((cal.get(Calendar.MONTH)+1)+"/"+cal.get(Calendar.YEAR);
            Det_Time=cal.get(Calendar.HOUR)+":"+
                    (cal.get(Calendar.MINUTE))+:"+cal.get(Calendar.SECOND);
            UDP_k = 0;
            InputForDecision obj = new InputForDecision();
            obj.setAttack("UDP Flooding Attack");
            obj.setDet_pack(udp);
            obj.setDet_Date(Det_Date);
            obj.setDet_Time(Det_Time);
            obj.setStart_Date(InputForStatistic.getUdp_Date());
            obj.setStart_Time(InputForStatistic.getUdp_Time());
            sendMessageDecision(obj); }
            UDP_count++;  
            beta = (float) 2 / (UDP_count + 1);
            UDP_mean=(int)((UDP_mean * beta) + ((1 - beta) * udp_no_of_packets));
    }

4. Decision Agent: Decision agent plays important role in anomaly agents. It receives detected attack message from statistic agent. It is the agent responsible for decision-making when attacks are detected; and depending on their protocol stored that attack into the database. Accordingly, type of packet and detected type, the attack is defined. They
send simple notification messages to the system administrator, and all results to decision support system to generate the charts and reports for that. When the alarm condition is satisfied and counter equal to $k$ then statistic agent send information to decision agent. Decision agent received that information and raised alarm as well as store information into a database and sends information to DSS. Decision agent raise dialog box in which user gets information about an attack and takes action on it, Figure 5.13 illustrates anomaly alarm dialog box.

![Figure 5.13](Illustrates Anomaly Alarm Dialog Box)

The implementation for decision agent is as follows: when statistic agent sends a “UDP flooding attack” along with packet information these all information is stored into the database. In the following code `DB.storeAnomalyDetection(d_id,protocol, obj.getStart_Time(), obj.getStart_Date(), obj.getDet_Time(), obj.getDet_Date(), s, det_pack,temp)` stored the attack information into the database.

```java
if (s.equals("UDP Flooding Attack")) {
    String protocol="UDP";
    DB.storeAnomalyDetection(d_id,protocol, obj.getStart_Time(),
    obj.getStart_Date(), obj.getDet_Time(),obj.getDet_Date(), s, det_pack,temp);
}
```

After storing into the database this information is sent to DSS. DSS is responsible for creating different types of charts and reports depending upon the information of detected packet stored into the database. The decision agent calls the `getAction();` method. This method for transferring detected type of attack to the database table. According to the type of attack and protocol type the data is send to DSS, So that it could be possible for DSS to analyze the data to display the charts on reporting tools.
5.5.2 Performance of Adaptive Threshold Algorithm with Multi-agents

Adaptive threshold algorithm is an extremely simple and generic algorithm that can be used for detecting the anomaly as per the deviations in the network. This algorithm based on the statically analysis. The statistical behavior is depended upon the network traffic, and detected attacks also this behavior varies according to time.

The algorithm takes this input from central agent and information agent but in NIDSMAM takes from packet sniffer. That mean, in our system actually gets the packets from network using the packet sniffer then it is processed and if it is not detected in misuse agents then it is transferred to anomaly agents for adaptive threshold algorithm. According to input data algorithm does some statistical and mathematical operations. This input data is processed, and from some past time experience we calculate threshold. If the threshold value accurate then such adaptive threshold mechanism can eliminate the need for user for threshold setting. Thus, NIDSMAM making more automated.

As a desired, adaptive threshold algorithm should also improve the accuracy of anomaly agents by tracking normal usage patterns and behaviors while such dynamic threshold is available for anomaly attacks. This will enhance the accuracy and performance of the system. If system adaptive threshold value goes beyond its desired value then NIDSMAM generates alarm. Figure 5.13 illustrates adaptive threshold algorithm (ATA) in anomaly detection agents.

![Figure 5.14 Illustrates Adaptive Threshold Algorithm (ATA) In Anomaly Agents]
From the above figure, we can explain how ATA used in anomaly agents. At very first block i.e. Packet information as input will come from the information agent then sensor agent will segregate the packet information according to the type of protocol. Description and implementation of ATA as follows:

\[ X_n \geq (\alpha+1) \times \text{mean}; \]

Where \( X_n = \text{no \_of \_packets} \) (here for TCP).

\[ \text{mean} = \left( (\text{TCP\_syn\_mean} \times \beta) + ((1 - \beta) \times \text{no \_of \_packets}) \right) \]

\[ \beta = \frac{(\text{float}) \ 2}{(\text{TCP\_syn\_count} \ + \ 1)}; \]

For example we consider the TCP_SYN. Algorithm first check and calculate the no. of packets, TCP_syn_mean of TCP_SYN then tcp_syn_k, means the no of consecutive time threshold is exceeded and then it is an attack, user can change TCP_syn_k at real time, and it is by default “3”. If \((\text{tcp\_no\_of\_packets} > \text{tcp\_syn\_threshold})\) satisfies means no. of packets are greater than the threshold value then we calculate the Alpha using \( \alpha = \frac{(\text{tcp\_no\_of\_packets} - \text{tcp\_syn\_threshold})}{\text{tcp\_syn\_threshold}}; \) . After that if \((\text{tcp\_no\_of\_packets} \geq (\alpha + 1) \times (\text{TCP\_syn\_mean}))\) i.e. Xn this condition is checked, and if that is true then we count the TCP_syn_k. Using TCP_syn_k we again check one more condition if \((\text{TCP\_syn\_k} == \text{tcp\_syn\_k})\), i.e. no of consecutive time threshold is exceeded. If this satisfies then algorithm generates the alarm “TCP SYN Flooding Attack”. This information is sending to decision agent.

5.5.3 The Role of Adaptive Threshold Algorithm in Statistic Agent
We present statistical anomaly detection algorithm with multi-agents for detecting flooding attacks which is the most common type of Denial of Service attacks. The use algorithm considered is an adaptive threshold algorithm for detected attacks with multi-agents. The evaluation performance achieves for using measurement performance (probability detection, detection delay, false alarm ratio, anomaly attack type, anomaly analysis, detected attacks anomaly, and anomaly detected) via using traffic of real time. On the other hand, how they are affected by the parameters of the algorithm and the characteristics of the attacks. Such an investigation can provide guidelines to effectively tune the parameters of the detection algorithm to achieve specific performance...
requirements in terms of the above metrics. In this work, we present and evaluate anomaly detection by using multi-agents with adaptive threshold algorithm for detecting flooding attacks. Adaptive threshold algorithm is a clear and simple algorithm that detects anomalies based on violations of a threshold that is adaptively set using recent traffic measurements. In particular the algorithm signals an alarm when the measurements exceed some threshold for a number of consecutive intervals $k$, where $X_n \geq (\alpha + 1) \bar{\mu}_{n-1}$ and $\mu$ is the measured mean rate.

We apply the adaptive threshold algorithm with multi-agents; we care to prove that a mere simple algorithm can display acceptable performance for some types of attacks. On the other hand to prove that algorithm based on multi-agents foundation can demonstrate sturdy performance over several attack types without necessarily being intricate and expensive to implement. In our situation, we note that the algorithm requires measurements of the aggregate number of packets and types of protocols in consecutive intervals of the time these measurements can be collected directly via multi-agents in anomaly detection agents. The value of the threshold is set adaptively based on an estimate of the mean number of packets which is computed from recent traffic measurements. We are using an exponential weighted moving average (EWMA) of previous measurements: $\bar{\mu}_n = \beta \bar{\mu}_{n-1} + (1 - \beta)X_n$ where $\beta$ is the EWMA factor.

When apply the algorithm above directly would yield a high number of false alarms (false positives). That can modification to improve performance is to signal an alarm after a minimum number of consecutive violations of the threshold. In this case, the alarm condition is given by:

$$\sum_{i=n-k+1}^{n} (x_i \geq (\alpha + 1) \bar{\mu}_{n-1}) \geq k$$

then alarm signaled at time $n$

Where $k > 1$ is a parameter that indicates the number of consecutive intervals the threshold must be violated for an alarm to be raised. For $k$ we by default set 3 users can change the value at runtime. We set threshold by some pass time experience and provide information to the user and the user set that threshold value. Because of that if the number of detection it is high then we can set $k$ value so, we can reduce false alarm. If an alarm condition is convinced than we increment counter by “1” then we checking with $k$
value if it is equal to $k$ we will pass information to decision agent if counter not equal to $k$ then we will not send information to decision agent.

The tuning parameters of the above algorithm are the amplitude factor $\alpha$ for computing the alarm threshold. The number of successive threshold violations $k$ prior to signaling an alarm the EWMA factor $\beta$ and the length of the time interval over which traffic measurements (number of packets) are taken.

In NIDSMAM they are several measurements for an estimate adaptive threshold algorithm: Alpha is the value calculated using $(\text{no_of_packets} - \text{threshold})/ \text{threshold}$; parameter alpha is tuning parameter by chosen value after every particular time interval, due to the interval time of delay detection. We show how to calculate alpha for every particular time interval for every protocol:

$$\alpha_n = \frac{X_n - T_n}{T_n} \quad n=1,2,3,..................... \tag{1}$$

Where $\alpha > 0$ and $X_n > T_n$

$n$ : number of intervals time

$X_n$ : no. of packets

$T_n$: value of threshold

And we can calculate alpha for one hour for every protocol:

$$\alpha = \frac{1}{n} \sum \alpha_n \quad \tag{2}$$

We can calculate beta by using the formula $[2/ (\text{count} + 1)]$. We show how to calculate beta for every particular time interval for every protocol:

$$\beta_n = \frac{2}{X_n + 1} \quad \tag{3}$$

And we can calculate beta on one hour for every protocol:

$$\beta = \frac{1}{n} \sum \beta_n \quad \tag{4}$$

We can calculate the mean by using the formula for example TCP SYN: $((\text{TCP_syn_mean} \times \beta) + ((1 - \beta) \times \text{no_of_packets}))$ for adaptive threshold
algorithm. We show how to calculate mean after every particular time interval for every protocol:
\[
\mu_n = ((\mu_{n-1} \cdot \beta_n) + (1 - \beta_n) \cdot X_n) \quad \text{........................................... (5)}
\]

\(\mu_n\) : computed over some past time window every particular time interval
\(\mu_{n - 1}\) the mean rate estimated from measurements prior to \(n\)

And after we can calculate mean rate for one hour for every protocol:
\[
\bar{\mu} = \frac{1}{n} \sum \mu_n \quad \text{........................................... (6)}
\]

In anomaly detection we are detecting attacks using adaptive threshold algorithm in this algorithm the value of threshold is very important. Because detection is purely depended on the value of the threshold. Using some past time experience we calculate the value of the threshold. We selected different values for threshold in interval time shows the appropriate detection threshold for system benign behavior in real time. We can select value of threshold to improve the accuracy the system by tracking unusual behavior.

We investigate the performance of the adaptive threshold algorithm for detecting (TCP SYN, TCP RST, TCP REST, UDP, and ICMP) flooding attacks. The performance standards include the measurement performance (probability detection, detection delay, false alarm rate, anomaly analysis, anomaly attack type, detected attacks anomaly, and anomaly detected), and how the parameters of the detection algorithm and the characteristics of the attack affect the performance.

In our system, experiments used real time traffic to capture packets containing pernicious activities prove that we can get precision results by adapting the detection threshold of anomaly detection agents on the basis of the observed input characteristics. In chapter six observed more analysis the results about the measurement performance and the characteristics of the attack how can affect the performance of the system by DSS.
5.6 INTERACTION BETWEEN MULTI-AGENTS USING JADE

JADE is a multi-agent platform that implements the basic services and infrastructure of a distributed multi-agents application. Jade is a suite of graphical tools that permits administrating and monitoring the activity of running agents. Following Figure 5.15 shows that, user interface for Jade agent named “rma”. Jade library provides this GUI based agent manager. In this manager each running instance of the Jade runtime environment is called container as it can contain several agents. A group of active containers is called platform. A main container always be active in platform so that all other containers register them self to the main container as soon as they start. Through this GUI we can easily monitor different agent’s activities and their communication through messages.

5.6.1 Interaction between all Agents in Misuse Detection Agents

In misuse detection agents we are checking the known attacks. In misuse detection agents there are (Detection agent, Central agent, Conversion agent, Response agent, UI agent, Alarm agent and Registration agent). The following Figure 5.16 shows initialization of agents in misuse detection agents. All agents are need to be initialized, and this is held by the `setup()` method. This method is called by each agent in misuse detection agents. Once the initialization process completes then communication starts with sending the
messages between these multi-agents. The arrows shown in Figure 5.16 indicate that warm communication is between these agents.

**Figure 5.16 [Initialization of Agents in Misuse Detection Agents]**

The misuse detection agents start detecting the attack after the initialization all the agents. In the following Figure 5.17 explains how the interaction between agents and communication happens to detect attacks in misuse detection. If any attack detected by our misuse agents then information is passed through next agent. all agents will interact with them, and if misuse does not detect any attack then it is forwarded to the anomaly detection agents. So, the arrow direction is only up to the central agent.

**Figure 5.17 [Detected Attack in Misuse Detection Agents]**
5.6.2 Interaction between all Agents in Anomaly Detection Agents

In anomaly detection agents contains mainly (Information agent, Sensor agent, Statistic agent and Decision agent). By using our “rma” tool of Jade we are showing in the following Figure 5.18 initialization of agents in anomaly detection agents. Starting i.e. Initialization process of these agents is carried out through the code and in that code we are using setup() method for each agent. This setup() method actually activates the each agent so that their communication starts for anomaly detection agents. After the initialization of all agents the agents are ready for detection of attacks in anomaly agents, following Figure 5.19 below shows that. It is clearly, we can see this arrow direction at first instance of communication. When attack detects then arrows are straight way i.e. one agent to other agent but only difference is that there are two arrows from statistic to decision agent. It means that attack is detected.

![Figure 5.18 Initialization of Agents in Anomaly Detection Agents](image)

The statistic agent sends packet information along with attack information to the decision agent. Also, the second instance of agent communication if an attack is not detected by the anomaly detection agents then arrows direction are moving forwarded.
5.7 DECISION SUPPORT SYSTEM (DSS)

Decision Support System is the computer application which is used for analyzing the information and data, so that the user can make decisions very simply. The purpose of software based DSS to help user for compiling information from raw data, documents, and various charts. There are three fundamental components of DSS first the database, second the model (decision context called user criteria) and third the user interface (charts or graphs). In NIDSMAM we are using data driven decision support system. Data-oriented DSS emphasizes access and manipulation of a time series of internal packet data. Figure 5.20 explains DSS in NIDSMAM.

Figure 5.20 [Illustrates (DSS) In NIDSMAM System]
The above figure shows “Decision Support System used in our system”. It includes five major components. DSS works according to the knowledge base i.e. database of NIDSMAM. DSS is one of the core and important part of NIDSMAM. We explain the above components of figure one by one.

5.7.1 Data Collection
DSS pyramid starts with data collection, in NIDSMAM we collect the data and information about packets using misuse agents and anomaly agents. This data collection is stored on our system’s database. The database contains the various tables like `master_packet_info`, `anomaly_detected`, `misuse_detected` etc. These tables contain information about the detected packets by misuse and anomaly detection.

5.7.2 Extract, Transform and Load
After the collection of data, DSS processes the information according to the results required for user. This extraction of data is held by the firing database queries. By firing the queries whatever results come that is not in well format because it is raw data information. Here, we get attack information (for both misuse and anomaly) but which is raw information. This information is processed and transformed to the required or desired format through code for loading to analysis part.

5.7.3 Analysis
Decision Support System passes the loaded information for analysis. Analysis part checks the attack information and analyze; it means removes the unwanted information checks and calculated exact results and relevant information. This analysis part gives us the more structural information about the type of attacks found in the misuse agents or anomaly agents. This is in the format required to the next part so, analysis is more important because the output of this part will be given to the reporting tools.
5.7.4 Reporting Tools

The information is analyzed and corrected then it is given to the reporting tools. In NIDSMAM we are using JFree charts tool for displaying various charts according to user requirement. These charts are parts of DSS and integrated with NIDSMAM. According to type of attack and the JFree chart structure charts will be generated.

5.7.5 User-Dashboard

User interface is generated by using the Java swing component. This is code is written in the java language. So the results of JFree charts are mapped with this UI component. User-Dashboard is responsible for displaying the output of JFree charts. According to user instruction which chart is going to display for which type of attack.

The following code explains about DSS working for anomaly detected module in anomaly agents for the last hour. As already, mentioned the first level of DSS module is for data collection part, in this part the data is stored at our database i.e. In case of this example the data is stored on anomaly detected table. The second level is for extracting the data by firing the various database queries these queries fetch the result about the last hour data i.e. first line of given code. By calling:

```
"data= ChartDB.anomalyDetectedHr(currentHourTime, prevHourTime, "TCP_SYN")
".
```

Whatever results will be transformed in a particular format as an input. Formatted result will be loaded for analysis. The analysis part will removes the unwanted information like irrelevant data in result set. After the reporting tool i.e. JFree chars are used to show the graph of anomaly detected for anomaly agents. We call this chart by calling “TCP_SYNAnomalyDetectedChart dataSet(data);”. This JFree tool will generate the chart according to the data which is passed to this function. After the generation of the chart, chart will be displayed on to the User-Dashboard. In DSS for UI we are using the Java Swing component. This Swing component having the panel component. We can see the code here:

```
JPanel pan = TCP_SYNAnomalyDetectedChart.createDemoPanel();
jPanel19.add(pan, 0);
```
These methods are called. Lastly the user gets the DSS output. Accordingly user can make decisions.

```java
Calendar now = Calendar.getInstance();
String currentHourTime = "" + now.get(Calendar.HOUR) + ":" +
now.get(Calendar.MINUTE) + ":" + now.get(Calendar.SECOND);
now.add(Calendar.HOUR_OF_DAY, -1);
String prevHourTime = "" + now.get(Calendar.HOUR) + ":" +
now.get(Calendar.MINUTE) + ":" + now.get(Calendar.SECOND);

try {
    ArrayList<HashMap<String, String>> data = new
    ArrayList<HashMap<String, String>>();
    try {
        data = ChartDB.anomalyDetectedHr(currentHourTime, prevHourTime, "TCP SYN");
    } catch (Exception ex) {
        Logger.getLogger(DEMO.class.getName()).log(Level.SEVERE, null, ex);      
    }
    if (jPanel19.getComponentCount() > 0) {
        jPanel19.remove(0);
    }
    TCP_SYNAnomalyDetectedChart.setDataSet(data);
    JPanel pan = TCP_SYNAnomalyDetectedChart.createDemoPanel();
    jPanel19.add(pan, 0);
} catch (Exception ex) {
    Logger.getLogger(DEMO.class.getName()).log(Level.SEVERE, null, ex);   
}
```

5.8 INITIALIZATION OF PROCESS IN NIDSMAM

When NIDSMAM starts the flash screen executes some major processes are carried out for NIDSMAM initialization. These processes are very important because outputs of these processes are used for detection of attack and intrusion detection etc. In NIDSNAN this flash screen is one kind of important loading components for NIDSMAM. Figure 5.21 illustrates NIDSMAM flash screen.
The following are the processes takes place in the background of above flash screen.

- There are mainly four processes for flash screen:
  1. Getting the LAN addresses
  2. Conversion of Snort rules to Jess rules format.
  3. Reading the database for misuse attack type.
  4. Fetching all source ports and destination ports from all_rules.clp file.

5.8.1 Getting the LAN Addresses

When flash screen executes the method "public ArrayList<String> getLanAddresses()" has been invoked by flash screen. This following method definition code explains us that, it calls the another method "Process p= Runtime.getRuntime().exec(command);" which executes the command and gives all LAN addresses in to a LanAddresses.txt.

```java
public ArrayList<String> getLanAddresses() {
    String[] command = new String[4];
    command[0] = "cmd";
    command[1] = "/C";
    command[2] = "arp -a >";
    command[3] = "D:\LanAddresses.txt";
    Process p = Runtime.getRuntime().exec(command);
}
```

The above code is for executing the command for fetches the LAN addresses into file Lan Addresses.txt. Once the LanAddresses.txt made then we read by the following code and parsed into the Array List which contains all LAN IP-addresses.

```java
FileInputStream fstream = new FileInputStream("D:\LanAddresses.txt");
while ((strLine = br.readLine()) != null) {
    k++;
    if(k>3) {
```
After getting all LAN addresses in to Array List we convert the source and destination IP address of incoming packets sniffed by packet sniffer, if the incoming packet’s source or destination IP-address is from LAN addresses i.e. from Array List then we replace that IP with “$HOME_NET” otherwise with “$EXTERNAL_NET”.

5.8.2 Conversion of Snort Rules into Jess Rules Format

For conversion of Snort rules to Jess rules format, we download the Snort rules from Snort website. Thus, following method ‘private static String ConvertIntoJESSRule(ArrayList<String> separateParts)’ is responsible for converting the snort rules to Jess rules format. Each Snort rules file is read by this method and converts to Jess rules format file called ‘all_rules.clp’. The following code explains Jess rule format.

```java
private static String ConvertIntoJESSRule(ArrayList<String> separateParts) {
    String j_rule = "",
    try{
        j_rule = "(defrule Rule" + counter + " " + quote + singlefile.getName().toString() + quote + " (protocol " + quote + separateParts.get(0) + quote + " (source_ipaddr " + quote + separateParts.get(1) + quote + " (source_port " + quote + separateParts.get(2) + quote + " (direction " + quote + separateParts.get(3) + quote + " (destination_ipaddr " + quote + separateParts.get(4) + quote + " (destination_port " + quote + separateParts.get(5) + quote + " (content " + quote + separateParts.get(6) + quote + " (ID " + separateParts.get(7)
```
5.8.3 Reading the Database for Misuse Attack Type

After conversion of snort rules to Jess rules control comes to reading the database of the alert message and type of attack; these two fields are maintained in \texttt{Attack\_File.txt}. These two fields are used in misuse detection module. When any attack found in misuse agents then it returns the alert message and Boolean true value. And this alert message is checked with alert type of our database. Thus, following method \texttt{`public HashMap\textless;String, String\textgreater; getAlertTypeAndTypeOfAttack()\texttt{`} returns hashmap which is used for pattern matching in misuse detection agents for type of attack.

\begin{verbatim}
public HashMap<String, String> getAlertTypeAndTypeOfAttack(){
    HashMap<String, String> alertAndAttackMap = new HashMap<String, String>();
    try{
        FileInputStream fstream = new FileInputStream("D:\Attack\_File.txt");
        while ((strLine = br.readLine()) != null)     {
            parts_of_str = strLine.split(\":\\);
            alertAndAttackMap.put(parts_of_str[0], parts_of_str[1]); }
    in.close();     }
    return alertAndAttackMap;   }
\end{verbatim}

5.8.4 Fetching all source ports and destination ports from all\_rules.clp file

The final step which is processed in the background of flash screen is getting all source and destination ports from \texttt{all\_rules.clp} file i.e. (Snort rules to Jess rules conversion). We fetch every rule in the file and get the source port and destination port put them into the Array Lists. The following code snippet only explains about the source ports but the logic required for destination ports is same as that of source ports. \texttt{sourcePorts.add();} method from the code is for adding that port number to Array List.
public ArrayList<String> getSourcePorts() {
    ArrayList<String> sourcePorts = new ArrayList<String>();
    try{
        FileInputStream fstream= new FileInputStream("D:\Snort\rules\all_rules.clp");
        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        BufferedWriter out = new BufferedWriter(new FileWriter("D:\source_port.txt"));
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        BufferedWriter out = new BufferedWriter(new FileWriter("D:\source_port.txt"));
        while ((strLine = br.readLine()) != null)   {
            if(strLine.contains("source_port")&&strLine.contains("destination_port"))  {
                parts_of_str = strLine.split(" ");
                for(int i =0; i < parts_of_str.length ; i++) {
                    if(parts_of_str[i].contains("source_port"))   {
                        if(!parts_of_str[i+1].contains("any"))   {
                            String temp=parts_of_str[i+1];
                            temp=temp.substring(1,temp.length()-2);
                            if(!sourcePorts.contains(temp)){
                                sourcePorts.add(temp);
                            }
                    }
                }
            }
        }
    }
}
These Array Lists are used for pattern matching because the incoming packet from packet sniffer is not in Jess rule format. So for maintaining the incoming packet formats same as Jess rule format. Accordingly, if the packet comes to the system then we match its source and destination port with our Array List. If it matches then do nothing with that port number and if not then replace that port number with “any” keyword.

5.9 LOGIN SCREEN NIDSMAM
Login screen will displayed after executing the flash screen. Here,NIDSMAM completely loads and asking for login authentication. We see the following Figure 5.22 snapshot for NIDSMAM login screen user need fill three fields i.e. his/her user name, password and select the interface. These fields are maintained into the database for valid or authenticated user for system.
Following code snippet explains the how NIDSMAM authenticate the user with password. Code actually matches the username and password entered by the user with database values which are maintained `admin_info` table.

```java
String mquery = "SELECT username,password FROM admin_info";
ResultSet maxres = st.executeQuery(mquery);
while(maxres.next()) {
    String user=(String)maxres.getObject(1);
    String pass=(String)maxres.getObject(2);
    if(username.equals(user) && password.equals(pass))
        flag=true;
    return flag;
}
if(db.Admin_login(jTextField1.getText(), jPasswordField1.getText()))
    {    new MainScreen().setVisible(true);
        pass=jPasswordField1.getText();
        user=jTextField1.getText();
    dispose();   }
else  {
    JOptionPane.showMessageDialog(null,"USERNAME password does not match");  }
```

User need to select the interface which is nothing but network interface. There are different types of interfaces available through which a computer is connected to the network. This network interface can be anything i.e. it may be LAN card, Virtual LAN card or Bluetooth network interface or WIFI network. Through this interface NIDSMAM understand that which network packet data would be sniffed. This interface is provided
from a packet sniffer, packet sniffer simply captures all of the packets of data that pass through a given network interface; we used Java as a platform. We need a tool which uses a packet sniffer and gives output the packets from the network. JPcap is a tool which used in Java for capturing network packets. Also, there is one tool for windows 32 bit called WinPcap for capturing the packets on windows machine.

**5.10 CHAPTER SUMMARY**

In this chapter, we explain the design hybrid NIDSMAM depended on multi-agents, using Jess, Snort rules, Adaptive threshold algorithm and DSS. We introduced system features and capabilities, also requirements for NIDSMAM design. We presented architecture of NIDSMAM using multi-agents. We review the role and concept of misuse detection agents includes [detection agent, central agent, analysis agent includes (conversion agent, response agent, alarm agent, UI agent, registration agent)], also explain the role and concept of anomaly detection agents includes [information agent, sensor agent, statistic agent, decision agent]. We explain the performance of adaptive threshold algorithm in statistic agent. We introduced interactions between multi-agents for misuse agents and anomaly agents by using Jade. We explain DSS, there are three fundamental components of DSS, first the database (or knowledge base), second the model (decision context called user criteria) and third one user interface (charts or graphs). DSS includes five major components [data collection, (extract, transform and load), analysis, reporting tools, user dashboard]. Finally, we presented initialization of process in NIDSMAM. There are mainly four processes for flash screen: getting the LAN addresses, conversion of Snort rules to Jess rules format, reading the database for misuse attack type, fetching all source ports and destination ports from all_rules.clp file.