41

INTELLIGENT IP PACKET FILTERING

M. Hashem[1], A. Mohamed[2], M. Wahib[3]
[1]*Vice President of A. D. Research and Development Center.*
E-mail: Mohamedhashem77@hotmail.com
[2]*Network & Distributed Sys. Dept. IRI, Mubarak City for Scientific Research and Technology App.* *E-mail: amohamedm@hotmail.com*
[3]*Faculty Of Computers and Informatics, Suez Canal University*
E-mail: wahibium@hotmail.com

ABSTRACT: Offering a moderate level of security at a lower cost requires the usage of packet filtering as a tool to provide network security. Securing system effectively requires the network administrator to well configure the packet filter with a thorough understanding of its capabilities and defects. Relying on the administrator in configuring and operating the packet filtering system might generate various threats due to human-error propabilities, so it's recommended that the system should be operated in an intelligent manner to automatically react to bad configurations such as duplications and contradictions, that decrease the proper matching time of the packet, with any of the predefined rules. Through out this paper, we propose an intelligent packet filtering technique. The proposed method enhances the performance in measures of individual packet filtering time. The method also includes a preprocessing algorithm for the rule organizing and removing of redundant rules. The proposed method gives the capability of auto-generation of rules for packets that do not match with any rule by the use of a rule-based expert system that uses the previously defined rules in deducing new rules. This paper describes traditional packet filtering and the rule-based Expert system. Ultimately, the paper evaluates the performance of this technique based on results of a simulation.

1. INTRODUCTION

Packet filtering is a controlling access to a network by analyzing the incoming and outgoing packets and letting them to be *routed* or *dropped,* based on the headers of the packet. Packet filtering is a one of many techniques, for implemented satisfying modest security requirements[2]. The internal (private) networks of many organizations are not highly segmented;

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35586-3_46

M. A. Ghonaimy et al. (eds.), *Security in the Information Society*

therefore highly sophisticated firewalls perhaps are not necessary for isolating one part of the organization from another. However, it is prefered to provide a sort of protection for the internal network from the outside world. A packet-filtering is a very appropriate method to provide an isolation of one subnet from another[10]. However, it should be put into consideration that packet-filtering technique does not provide the same level of security as an application or proxy firewall. All the IP networks, except the most trivial of IP networks, are composed of IP subnets and contain routers. Each router is a potential filtering point because the cost of the router has already been absorbed, additional cost for packet filtering is not required.

Unfortunately, the existing packet filters proved many limitations and weak points (e.g. long matching time duo to the tabular structuring of rules and the necessity to manual intervention of the administrator on the incoming of a packet without a matching rule), which leave the administrator with little assurance that the traditional used filters are quietly and entirely specified. Current packet filters still require many improvements in order to offer a higher security level and to be operated in an intelligent manner.

The traditional packet filtering method, its process, main weak points and their effects on performance are discussed in section 2. The proposed rule-based Expert system packet filtering technique is investigated in section 3. A comparison of the two methods, regarding their performance and behavioral bases, is illustrated in section 4. Finally section 5 includes simulation results of the two methods, designed for measuring and comparing performance of the two methods.

2. TRADITIONAL PACKET FILTERING

2 .1. Introduction

In the context of a TCP/IP network, a packet filter watches each individual IP data-gram, decodes the header information of in-bound and out-bound traffic and then either blocks the data-gram from passing or allowing the data-gram † to pass, based on the contents of the source address, destination address, source port, destination port and/or connection status. This is based on certain criteria defined to the packet filtering tools. The leading IP routers, including Cisco, Bay, and Lucent can be configured to filter IP data-grams. Many operating systems can be configured for packet filtering. Support of packet filtering via *ipchains* is included by default in the Linux kernel. Windows NT and Windows 2000 support packet filtering. Virtually, all commercial firewalls support packet filtering. Some commercial firewalls also have the capability of filtering packets based on the state of previous packets (state full inspection) [1].

All Packet filters operate in the same basic fashion. They parse the header of a packet to extract the data needed to identify and handle the

packet. Then, apply rules from a simple rule table to determine whether to *route* or *drop* the packet. Header fields that are available to the filter are packet type (TCP, UDP,...etc.), source IP address, destination IP address, and destination TCP/UDP port. The filtering rules are expressed as set of entries for the rule table created by the administrator. That set of rules depends on the needs and requirements of the organization. The filtering rules appliance (the appliance here refers to the method by which a decision is taken from the reserved filtering-rule table) for the traditional technique is simple. The rules are just inspected one after another in a sequential search manner. The order of the rules is specified by the administrator in some systems. In others, the rules are reordered by the router, such as applying the rules referring to more specific addresses (such as rules pertaining to specific hosts) before rules with less specific addresses (such as rules pertaining to whole subnets and networks). Some routers with very rudimentary packet filtering capabilities don't parse the headers; but instead require the administrator to specify byte ranges within the header to examine as well as the patterns to look for in those range [10].

Generally, the filtering rules are expressed as a table of rules (containing conditions and actions) that are applied in a certain order until a decision to *route* or *drop* the packet is taken. When a particular packet meets all the conditions specified in a given row of the table, the action specified in that row (whether to route or drop the packet) is carried out. In some filtering implementations, the action can also indicate whether or not to notify the sender that the packet has been dropped (through an ICMP message), and whether or not to log the packet and the taken action.

2 .2. Packet Filtering Example

Rule	SrcAddr	DstAddr	Port	Action	Priotrity
A	135.79.0.0/16	123.45.6.0/24	80	route	300
B	135.79.99.0/24	123.45.0.0/16	80	drop	200
C	0.0.0.0/0	0.0.0.0/0	*	drop	100

Rule C is the default rule, applying these rules on the sample packets with respect to the rule priority, will generate the following results:

Packet	SrcAddr	DstAddr	Port	Action
1	135.79.99.5	123.45.0.10	80	*drop*(B)
2	135.79.0.5	123.45.6.10	80	*route*(A)

The example uses *rule* priority as bases for choosing the order of appliance, other systems might use other criteria.

† Throughout the context of the paper, *drop* will refer to the action of blocking the packet by the packet filter; and *route* will refer to passing the packet to destination subnet.

2 .3. Packet Filtering Limitations

The major problem with many current traditional packet filtering implementations, such as network security mechanisms, is that the filtering is usually too difficult to configure, modify, maintain, and test, which leaves the administrator with little confidence that the filters are quietly and entirely specified. In addition to the difficulty in configuring, traditional packet filters have more caveats as the problem of IP fragmentation which requires to keep history for incoming packets to avoid flooding and smurfing, especially the handling of start-of-connection packets.

Other problems are long matching time due to the tabular structuring of rules which requires manual intervention of the administrator on the incoming of a packet without a matching rule, the demand to remove *rule* redundancy and minimizing the *rule* set. These three problems are illustrated as follows:

• The first one is due to relying on the administrator in configuring and operating the packet filtering system. This manual operating by the administrator may generate various threats due to human-error propabilities. He may add a rule that is redundant (i.e. the newly added rule is a complete subset of another existing rule, and the priority of the existing rule is higher than the newly added rule, so the a redundant rule; and vice versa. If the existing is a subset of the new rule but with lower priority, then the existing rule will be the redundant one).

Example:

Rule	SrcAddr	DstAddr	Action	Priority
A	135.79.99.0/24	123.45.0.0/16	*drop*	50
B	135.79.0.0/16	123.45.6.0/24	*route*	30
C	0.0.0.0/0	0.0.0.0/0	*drop*	10

If a rule D {135.79.99.5/10, 123.45.0.10/12, *Deny*, 40} is to be added, it will be redundant because if a packet matches with rule D it surely matches with rule A (because D is a subset of A). As long as A has priority higher than D, then the packet will match with A; and rule D will never be used in this case. The previous assumption will be valid if A was subset of D and priority of A was less than that of D, but in this case A will be redundant.

• Some packet filters use a default rule for the system, desiring very high security as military sites implement the following policy "*Deny* every incoming packet unless a rule matches with that packet" by defining a default rule with the action *Deny* and having the least priority in the existing rules. Other systems, not having security as an extreme goal (e.g. A high school), implement the policy of "*Permit* every incoming packet unless a rule matches with that packet" using the default rule with action *Permit* with the least priority. Some packet filters don't implement any of the two policies, the packet just matches with the existing rules that don't hold a default rule. For such systems, if the packet doesn't match with any of the exiting rules, the program informs the administrator with the incoming of a

packet without a matching rule, and the decision is then lift to the administrator. The problem with this system is that it needs from the administrator to be on-line every time the firewall is up. Also the time taken for the administrator to decide what to choose will time out the inspected packet as well as the need to the physical existence of the administrator all the time the packet filter is up, in order to monitor the system.

- This method is very slow specially when the number of rules is large. (i.e. the requirements and needs of the organizations require many rules to be represented). As the large number of rules increases, the number of comparisons will create an over load on the machine. The tabular structure of the *rule* set needs to be replaced by more efficient structure to minimize the matching time.

A proposed method to overcome these problems is to use *Intelligent Packet Filtering using a rule-based expert system* which is a packet filtering program that is operated in an intelligent manner to automatically react to configure faults as duplications and contradictions, handling packets, that don't match with any of the predefined rules as well as minimizing matching process time.

3. EXPERT SYSTEM BASED PACKET FILTERING

3 .1. Expert System Overview

Expert systems are part of a general category of computer applications known as artificial intelligence. Designing an expert system needs a knowledge engineer, an individual who studies how human experts makes decisions and translates the rules into terms that a computer can understand [5]. Expert system is defined as: A computer application that performs a task that would additionally be performed by a human expert. For example, there are expert systems that can diagnose human illnesses, make financial forecasts, and schedule routes for delivery vehicles. Some expert systems are designed to take the place of human experts; meanwhile others are designed to aid them† [7]. The main components of the Expert system are shown in the Figure.1

† Throughout the part discussing the Expert system method, *rule* refers to the filtering rules added by administrator and RULE will refer to one of the rules in the rule-base.

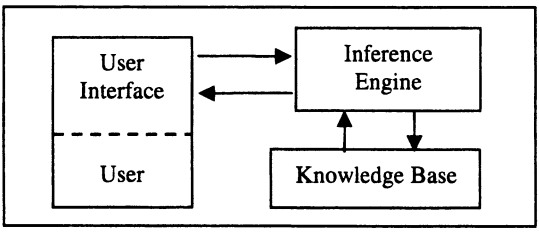

Figure 1(General Block Diagram of an Expert System)

3 .2. How Expert System Based Packet Filtering Works

In the traditional method, adding the rules, determining the sequence of appliance of rules and learning all are done manually by the administrator.

This may cause the existence of duplication and unnecessary rules. It also needs the administrator to be on-line every time, as long as the firewall is up, to add the new rule resulting from the incoming of packets, that don't match any rule which is very hard especially for firewalls, that run all the time. This will also be a drawback in the speed of the firewall.

So we need a firewall that is adaptive to unhandled situations, and can run the learning process internally without the administrator. The administrator will just insert all the rules initially (depending on the requirements and needs of the organization) and won't need add any rules later. Another improvement is the removal of redundant *rules* that will just maximize the size of the *rule* set without using it. The last point to be done is structuring the rule in a tree structure to minimize the matching time and aiding in the design of the Expert System. See Figure2

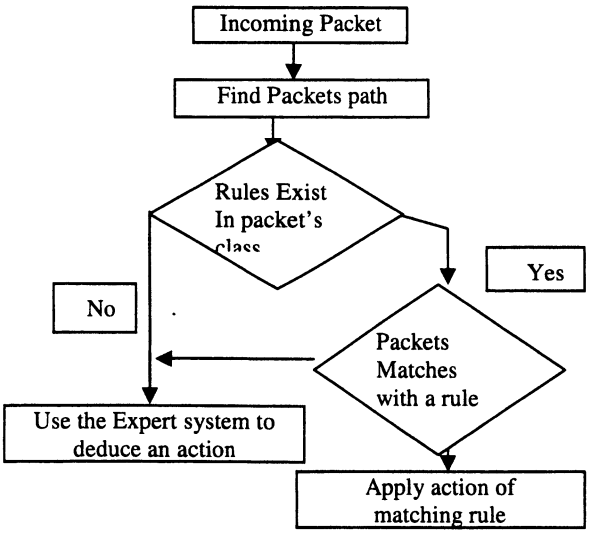

Figure 2 (Flow Chart of Expert System packet filtering)

3 .2 .1. Rule structuring:

Initially, All the possible port numbers are subdivided into 4 classes. Such that class holds a range of ports (could be applicable for any number of

classes but more than 4 classes will increase the complexity). The ranges of classes are not equal because the *rule* won't have equal and same probabilities to fall in one of the classes with equal ranges. As most of the packets port numbers fall inside the range 0-1023 which are the reserved ports used by most applications; so three classes will cover the range of the reserved ports; and one class only will cover the other ports. The next field in the rule (IPSrc) is also divided in a similar way into 4 classes (equal ranges for the previous .) Such that Port class forks into the four IPDest classes. Then the remaining field IPSrc is similarly divided into four classes (also classes with equal ranges). Such that IPSrc classes are the third level in the tree representing the Domain. Time is ignored in this method because it doesn't have fixed boundaries to be divided into several classes; and now our domain (all valid values for the fields of a rule) is represented in a tree hierarchy; and each *rule* will have one or more paths. The next algorithm describes the steps for adding a new *rule.*

The Algorithm

```
1.   NewRule.Path  = CreatePath( MyPortClass,  MyDestIPClass, MySrcIPClass).
2.   RemainingRules = Intersect(NewRule.Time, FamilyRules.Time).
     if ( IsEmpty(RemainingRules) )
           Add(NewRule), Return
       //Determine which rules intersect with the new rule in the time
3.   RemainingRules = Intersect(NewRule.Port, RemainingRules.Port).
     if ( IsEmpty(RemainingRules) )
           Add(NewRule), Return
     //Within the previously intersecting rule, check which of them are intersecting in Port
4.   RemainingRules = intersect(NewRule.DestIP, RemainingRules.DestIP).
       if ( IsEmpty(RemainingRules) )
           Add(NewRule), Return
       //Intersect again with remaining rules but in the DestIP Field.
5.   RemainingRules = Intersect(NewRule.SrcIP, Remaining.SrcIP).
       if ( IsEmpty(RemainingRules) )
           Add(NewRule), Return
6.   for I = 1 to Num(RemainingRules)
          if(NewRule subset RemainingRules[I] )
              if(RemainingRules[I].Priority GREATER NewRule.Priority)
                   Return
       //Don't add the new rule, it's redundant.
7.   for I = 1 to Num(RemainingRules)
         if(RemainingRules[I]  subset NewRule)
              if( NewRule.Priority GREATER  RemainingRules[I].Priority )
                  Delete(RemainingRules[I])
                    Add(NewRule), Return
       //Delete redundant rule and add the new rule.
8.   Add(NewRule), Return
```

Step 6 of the algorithm handles repeated *rules.* If the new *rule* is a subset of at least one *rule* (which means that the new *rule* completely falls inside another rule), then don't add the *rule* and notify the user that the similar *rule* already exists. If the Action of new *rule* is not like the action of the other *rule* or the priority of the new *rule* is higher than that of the other

rule, then the new *rule* is added. Step 7 handles the case when the new *rule* contains at least one *rule* (which means that an existing *rule* completely falls inside the new *rule*), then the new *rule* is added; and the other *rule* is deleted. The added rule will be given the priority of the deleted rule (if it is more than the priority assigned to the new *rule* by the administrator). If more than one *rule* will be deleted in this step, the new r*ule* is given the highest priority of the deleted *rules*. If a rule falls inside more than one family, the rule will have many paths (family is set of rules with same path); and the algorithm will be repeated for each family. *Rules* covering wide ranges will increase the number of rules in families because the *rule* will have more than one path and will be a member in more than one family; but anyway the number of *rules* in each family will be much less than the number of the whole set *rules.* Also wide ranges benefit in eliminating many rules because many *rules* will fall inside the wide range *rule* and thus there is a big chance for those to be deleted.

3 .2 .2. Learning a new rule

A simple rule-based expert system is to produce new *rules* for unexpected packets, depending on the relation between the packet and it's neighbors in the family. Also depending on occurrence of similar port numbers and IP addresses.

The priority of fields to check is ordered as follows: Port, DestIP, SrcIP. The first 2 RULES depend on occurrence of the port numbers in the existing *rules,* RULES 3 and 4 are the same as the previous but uses the DestIP then the port. RULES 5 and 6 are the same but for SrcIP port then DestIP. RULES 7 and 8 inspect the *rules* in the packet's family and take decisions depending on the distance between the packet and *rules.*

Example: RULE2:

IF (SimilarAllActions (RulesWithPort(packet.Port))
AND ActionIs(ACCEPT)
AND someRulesActionWithDestIP(ACCEPT))
THEN >> *ConstructNewRuleFromPacket(ACCEPT)*

The RULE states that if all the *rules,* with port similar to port of incoming packet, have action ACCEPT, and within the rules with same port not all *rules,* with same DestIP, have action BLOCK; then create a new *rule* for this packet with action ACCEPT.

RULE8:

IF (DistanceBetweenPacketWithRulesOFActionACCEPT
LESS THAN DistanceBetweenPacketWithRulesOFActionBLOCK)
THEN>> *ConstructNewRuleFromPacket(ACCEPT)*

The RULE states that when the distance between the packet and *rules* with action ACCEPT (*rules* in the family of the packet, not any rule) more than the distance between packet and remaining *rules* in family with action

BLOCK; then create a new *rule* for this packet with action ACCEPT. Otherwise, creates the new *rule* for this packet with action BLOCK.

Identifying the *rules* with a field equal to that of the packet will be simpler, using the tree containing the *rules*. The *rules* with that particular field will be contained in one or more sub-trees (1 for the port, 4 for DestIP, 16 for SrcIP). The Distance in RULES 7 and 8 is between the packet and a *rule,* the distance is measured as following:

PortDistance = DestIPDisatance = SrcIPDisatance = 0
IF(packet.port OUTSIDE range(rule.port))
PortDistance=Min(|packet.port-LowestBound(rule.port)|, |packet.port-HighestBound(rule.port)|)
IF(packet.DestIP OUTSIDE range(rule.DestIP))
DestIPDistance=Min(|packet. DestIP-LowestBound(rule. DestIP)|
, |packet. DestIP-HighestBound(rule. DestIP)|)
IF(packet. SrcIP OUTSIDE range(rule.SrcIP))
SrcIPDistance=Min(|packet. SrcIP-LowestBound(rule. SrcIP)|
, |packet. SrcIP-HighestBound(rule. SrcIP)|)
Distance =PortDistance^2/100 + DestIPDistance/2^10 + SrcIPDistance/2^20.

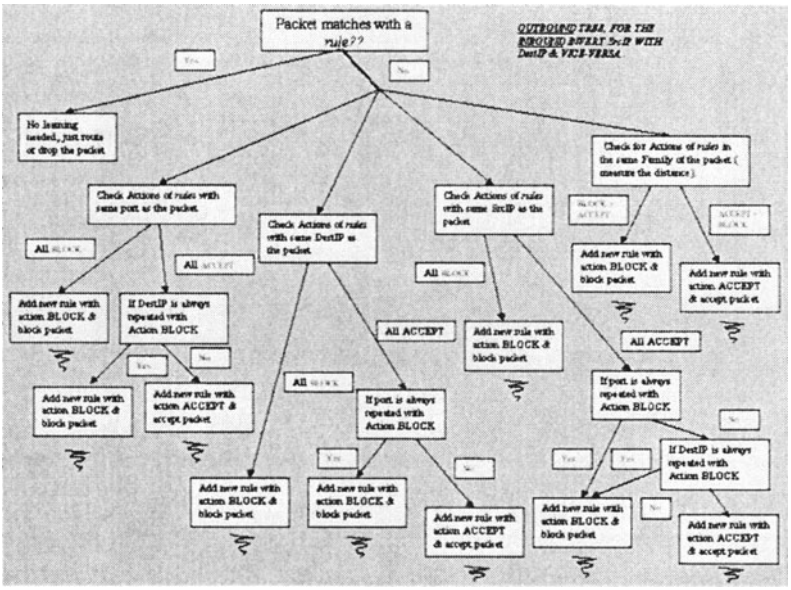

Figure.3 holds a part of the tree used in the inference algorithm.

The inference Algorithm:

1. For all rules … IF((packet.port=rule.port AND rule.Action=BLOCK)
OR ((packet.port=rule.port AND rule.Action=ACCEPT)
AND NOT(AllRulesWithDestIP(packet.DestIP).Action=ACCEPT)))
THEN>> AddNewRule(BLOCK)
ELSE
IF (packet.port = rule.port AND rule.Action =ACCEPT
AND AllRulesWithDestIP(packet.DestIP).Action=ACCEPT))
THEN>> AddNewRule(ACCEPT)
2. For all rules … IF((packet.DestIP=rule.DestIP AND rule.Action=BLOCK)
OR (packet.DestIP= rule.DestIP AND rule.Action=ACCEPT

```
        AND NOT(AllRulesWithPort(packet.Port).Action=ACCEPT)) )
          THEN>> AddNewRule(BLOCK)
    ELSE
    IF (    packet.DestIP = rule.DestIP AND rule.Action =ACCEPT
      AND AllRulesWithPort(packet.Port).Action=ACCEPT)  )
          THEN>> AddNewRule(ACCEPT)
3.  For all rules ... IF(    ( packet.SrcIP=rule. SrcIP AND rule.Action=BLOCK)
     OR (    packet. SrcIP= rule. SrcIP AND rule.Action=ACCEPT
        AND NOT(AllRulesWithPort(packet.Port).Action=ACCEPT))
     OR (    packet. SrcIP= rule. SrcIP AND rule.Action=ACCEPT
        AND AllRulesWithPort(packet.Port).Action=ACCEPT
        AND NOT(AllRulesWithDestIP(packet.DestIP).Action=ACCEPT))) )
          THEN>>  AddNewRule(BLOCK)
    ELSE
    IF(    ( packet.SrcIP=rule.SrcIP AND rule.Action=ACCEPT)
     AND AllRulesWithPort(packet.Port).Action=ACCEPT
     AND AllRulesWithDestIP(packet.DestIP).Action=ACCEPT  )
          THEN>> AddNewRule(ACCEPT)
4.  DistanceWithBlock  = sum(Distance(packet,rule.Action=BLOCK))
    DistanceWithAccept= sum(Distance(packet,rule.Action.ACCEPT))
    IF(DistanceWithBlock<DistanceWithAccept)
          THEN>> AddNewRule(BLOCK)
    ELSE
          AddNewRule(ACCEPT)
```

3. Comparison

The comparison includes behavioral type in section 4.1, and performance type in section 4.2.

4 .1. Behavioral Comparison

The comparison is divided into three parts, 4.1.1 deals with the case of adding a new *rule* by the administrator. 4.1.2 handles an incoming packet with a matching rule. 4.1.3 handles incoming packet without a matching rule. This is the part that needs learning using the inference engine.

4 .1 .1. New rule added by administrator

The traditional packet filtering acts as follows: Adding the new rule means adding a new row in the table of rules. Some systems just append the rule to the existent set because the system enforces a particular order of rule application, based on the criteria in the rules, such as source and destination address, regardless of the order in which the rules were specified by the administrator.

Most systems ask the administrator for a priority number to be assigned to the rule in order to apply the rules in the sequence specified by the administrator until they find a rule that applies. In this type of packet filtering, the administrator must be careful when configuring the system because inaccurate sequence of applied rules may lead to undesired results and cause security holes in the system.

Expert system based packet filtering introduces the new *rule* to the algorithm in section 3.2.1 to construct the knowledge base on which our expert system is based on. The knowledge base here will be the *rules* added by the administrator.

This part of the Expert system method removes duplication rules and organizes the rules in a way to make it later easier for learning. This part will be preprocessing and that may add extra load to the system. However, adding the *rule* could be made off-line.

4 .1 .2. Incoming packet with a matching rule

The traditional packet filtering acts as follows. When a packet arrives, rules are checked with the sequence specified by the administrator until a rule matches. Then the action corresponding to that rule is applied on the packet by either *route* or *drop*. Here a little modification could be made to improve this way by matching the fields in an increasing ranges order of values. For example, the port number has much less possible values than the IP; so if it is matched first, it will reduce the number of comparisons made by mismatching rule.

Expert system based packet filtering proceeds as follows:

a.1) The path of the packet is determined to know to which family the packet belongs (Must be only one family).

a.2) The packet is matched with all the rules in the family in a way similar to the traditional method (applying one rule after another in a priority order until one rule matches. However, the number of comparisons obviously will be much smaller than those in the traditional method due to comparing to one family only not to all *rule* table) which results in either finding a matching rule, that decides either to drop, or to route the packet. If the packet doesn't match with any rule, the *Learning Routine* is invoked, which is a rule-based expert system described in section 3.2.2.

4 .1 .3. Incoming packet without a matching rule

The traditional packet filtering acts as follows: If the incoming packet doesn't match any of the rules in the table. This invokes the *Learning Routine,* that informs the user with the incoming of a packet, that doesn't match with any of the recorded rules. Then, the administrator has the ability to add a new rule to handle this type of packet. Accordingly, adding the new rule is done manually, which means that the learning rate for this method will be zero. *Expert system based packet filtering* for this case invokes the expert system in section 3.2.2. The administrator could view the action taken by the expert system first, to make sure that the action matches with the desired action.

4 .2. Performance Comparison

Traditional packet filtering is the safest method because the process of adding a new rule is always made by the administrator; so the ratio of

mistakes in the learning process is zero (the ratio of learning process here is concerned only of the number of new rules added, that meets the desires of the administrator but not concerned of whether the new rule will be effective or will cause a hole in the system). However, this method is very slow especially when the number of rules is large. (i.e. the needs of the organizations require many rules to be represented), as the large number of rules increase, the number of comparisons will create an over load on the machine. Therefore, this method won't be efficient in this case and will be a bottle neck. Also isn't applicable if no default *rule is* added and internal learning is desired because the administrator will need to monitor the system all the time. This method is recommended if the security is a major factor for the system; (e.g. Military usage) and the rules are limited in number.

Expert system based packet filtering is a highly efficient method in many ways:

1. It eliminates the duplication and unnecessary rules added by the administrator that may cause conflictions and increase the size of the *rule* set.

2. For the case of packets with matching rules, the number of comparisons is much less than in the traditional method. This is because the packet is filtered with the rules in the same family and not with the whole rule table. So even if a default *rule* is added by the administrator, the algorithm could be used to increase performance in terms of matching time.

3. The learning process is automated and doesn't require the intervention of the administrator. However, the learning process will be a draw back in time if the knowledge-base is of a large size (i.e. large *rule* set). The time consumed for the expert system to return a deduced action anyway will be much less than the time taken by the administrator to receive the notification of an unknown packet and to take a decision.

5. SIMULATION

A simulation is proposed to compare between the performance of the two methods, whose tools are:

1.Rule files and Packet files that are created statically with various sizes.

2.The Simulation program was executed on Pentium III 700 MHZ processor, 128 MB RAM using winNT version 4.0.

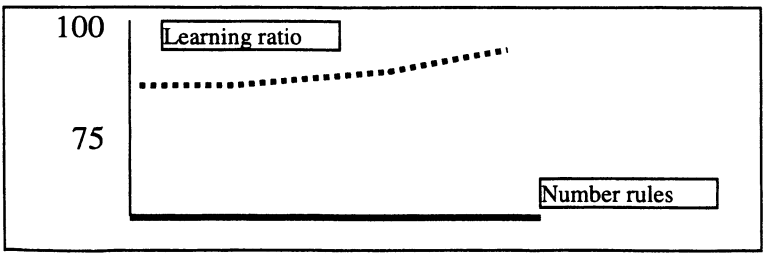

Figure 4 (Graph for performance regarding learning ratio)

As illustrated in Figure.4, the traditional method has zero learning ratio because the learning in this case is manual. Meanwhile, the expert system method had a ratio of 75%. This ratio is not within a limited range. Some rule sets may have 100% and others may be zero. It entirely depends on the administrator configuration and the requirements of the organization.

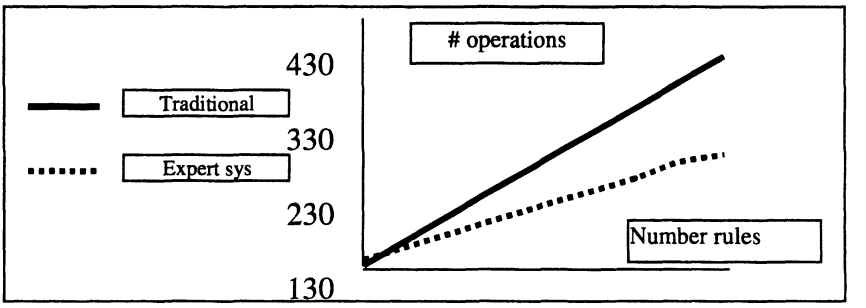

Figure. 5 (Graph for performance regarding number of operations)

From figure.5. the traditional method is linear as being O(n) like the sequential search. Expert system method's graph is semi-linear but much less in value than the traditional method.

6. CONCLUSION

Due to the limitations of the traditional packet filter in providing security and isolation for the system, we proposed an intelligent packet filtering technique.

The proposed technique enhances the performance compared to the traditional one. It also includes the capability of an auto-generation of rules for packets that do not match with any rule in the rule-based Expert system.

REFRENCES

[1] S. M. Bellovin, 1989, Security Problems in the TCP/IP Protocol Suite.
[2] D. Brent Chapman, 1992, Network (In)Security through IP packet filtering.
[3] Douglas E. Comer, 1991, Internetworking with TCP/IP.
[4] George Droffner, 1997, Neural Networks Techniques.
[5] Louis E.Frenzel.,1990, Crash course in Artificial Intelligence and expert systems.
[6] Vasant Hanavar, Leonard Uhr, 1994, Artificial Intelligent and Neural Network steps toward principal Integration.
[7] George Luger, William Subblefield, 1993, AI structures and strategies for complex problem solving.
[8] J. Quon, 1998 , Firewall complete reference.
[9] H. White, 1989 , Learning in Artificial Neural Network.
[10] G. Zammler., 1998 , Internet Security and Firewalls.
[11] Jacek M.Zurada, 1995, Introduction to artificial neural systems.