Journal of Computer Science 1 (2): 259-269, 2005
ISSN 1549-3636
© Science Publications, 2005

Artificial Neural System for Packet Filtering

[1]M.I. Buhari, [2]M.H. Habaebi and [2]B.M. Ali
[1]King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
[2]Department of Computer and Communications Engineering
University Putra Malaysia, Serdang 43400 Selangor–Malaysia

Abstract: In this study, we analyzed the use of neural network for packet filtering. The neural network system was designed in eight ways with input to the neural network in the form of either access rules or optimized access rules or binary form of access rules or representing wildcards as 0 and 255, or combination of them. These trained neural networks were analyzed for their correctness and the performance aspects such as training time using test data. In order to further improve the security, the data related to the local usage of the network was also used to train the network. An example of implementing these trained systems in active networks packet filtering was presented.

Key words: Packet Filtering, Back Propagation, Firewall, Active Network, TCP

INTRODUCTION

The firewall has to use an IP router to control the passing of any packet from the Internet into the Intranet. Packet filtering parses the headers of the received IP packets and forwards or discards the packets according to an Access Control List (ACL) specified by a network administrator. The performance of the whole router depends on the procedure on which the rules in the ACL are applied to the packet and a decision made to allow or reject the packet. Processing all packets passing through a router degrades the packet-forwarding performance. Also, the performance of the off-the-shelf routers is degraded in proportion to the complexity of the ACL. Hence, IP routers that forward packets more efficiently are to be explored.

As the performance of the firewall has been highly affected due to the process of sequential parsing, Miei *et al* [1] have proposed a compiler for parallelizing IP-packet filter rules, to improve the network security and reduce the degradation in packet-forwarding performance. In the proposed method there is no need for any intermediate program. The current day packet filtering is done using sequential parsing methods [1]. With the sequential parsing technique, increase in the number of rules in the ACL list will cause the parser to consume more time and become inefficient.

According to Murthy *et al.* [2], a firewall can be made of an external router along with a bastion host. A bastion host is the important component of a firewall that performs the tasks of user authentication, machine verification, logging of all security events to an internal host and the execution of proxy servers for all allowed services. The most serious drawback of this kind of system is that if the traffic through the host increases, the system might be overloaded. This can be avoided by using multiple hosts. The second drawback is that the user may feel that the system is too restrictive.

To overcome the drawbacks of bastion host (i.e., to reduce the overload on the firewall), Gittleson *et al.* [3] proposed the concept of smart filter. The smart filter reads the header of a session's first packet, compares it to the rules and if approved, routes successive packets through a cache. While each packet files through the cache, the filter compares its header to that of the first packet to verify that it belongs to the same session. Thus, it doesn't need to verify each packet against the rules.

Access Rules Implementation Using Various Techniques: Currently, packet filtering is done using access control lists, which are processed sequentially. IP packet filtering using optimized-sequential processing, neural network and expert systems was done and results are reported in Table 6. In each of the above three methods proposed, the rules are optimized and hence reduce the degradation in packet filtering performance. Time is an important criterion in the IP packet filtering. Even in the case of increased network traffic, the proposed methods can be efficiently employed for IP packet filtering. From Table 6, it is clear that out of all the three methods considered, the expert system has an edge over the other methods as it processes a fewer number of rules than the other methods. It is found to be better than the optimized sequential parsing and the neural network methods of packet filtering. But, security lapse was found with both the neural network and expert system implementation. In this paper, we try to improve the performance of neural network system by applying different techniques and then take care of the security lapse also.

Problem Motivation: Security lapse was noted for both neural network and expert system oriented packet filtering. To solve the security lapse, we analyse the correctness of the trained neural network oriented IP packet filtering and its performance impact. As the use of access control rules alone for neural network does not provide correct output as required by the packet filtering system, different neural network systems were designed and trained in the following ways:

* Using Access Control rules.
* Using Access control rules and replacing wildcards with 0 and 255.
* Using Access control rules in binary format.
* Using Access control rules and wildcards with 0 and 255 in binary format.
* Using Optimized Access Control rules.
* Using Optimized Access control rules and replacing wildcards with 0 and 255.
* Using Optimized Access control rules in binary format.
* Using Optimized Access control rules and wildcards with 0 and 255 in binary format.

As the number of inputs and in turn the architecture (number of input and hidden neurons) of the neural network with the above-mentioned methods varies, the training process of the neural network also varies. These trained neural network systems were tested for consistency with the actual action from the ACL rules described in Tables 1, 2, 3, 4 and 5. The selected test data is used to check the correctness of the trained neural network system. The security and performance aspects of the trained system were determined based on:

* Whether the system attained the maximum allowed error during the training process.
* The result of the trained system is compared with the action based on the ACL rules.
* Amount of time taken in terms number of iterations or epochs by the neural network system to train.
* Number of input needed to train the system along with the architecture of the network.
* Number of rules and its impact on the training of the system.
* Use of wildcards and how neural networks is affected by that use.
* Impact of optimization of the access rules and the conversion of the access rules into binary.

BACKPROPAGATION NETWORK APPROACH TO IP PACKET FITLERING

In an attempt to improve the performance of IP routers in forwarding packets, we propose to use the back-propagation oriented neural network algorithm to train the network to learn the ACL rules as demonstrated below.

Usage of Bakpropagation Network: The neural network algorithm proposed for this experiments is designed to be real-time trainable because any addition of ACL rules doesn't incur the repetition of the whole training process. The network sees the N most recent patterns through a shifting time window. It learns to predict the next value of the series. First, the network is trained to the point of convergence using a set of access rules previously designed. During this process, the weights are updated from a random start configuration to the values corresponding to the desired transfer function. Then, this function is inserted online and works well without further modification.

The disadvantage of a neural network is that it takes a long time for training depending upon the various sets of patterns provided to it. In the case of general neural networks, it is possible that the already trained set of data is affected by a newly arriving training pattern. In order to avoid this kind of discrepancy, a part of the output is fed back to the input of the network.

In the case of a back propagation network, the training phase needs both the input and their corresponding output. So, the network is made of varying inputs and one output. The number of inputs varies with regard to the different forms of input provided to the system, like binary or optimized. The "newff" (Feed forward back propagation) function was used to train the network and the trained network is simulated to identify the output for any specific input pattern.

If in one case, any one of the input parameters is absent then the wild-card character is used, which represents two input patterns-the lower (value as 0) and upper bound (value as 255) of the corresponding parameter. Only one output neuron is used to identify a permit or deny.

The weights are randomly chosen during initialisation of the network. The network is trained to give the same output for the two extreme values of any input, which is represented by wild-card character. This has been ascertained using the justification that the output for any intermediate input value will be the same as the output that it's got when the extreme values are used to train a system to attain a fixed value.

Neural Network Based IPv4 Filtering: Originally, as per the data available from the packet, the number of input parameters is twelve. These are Packet Type (One input), Source IPv4 address (Four inputs, standing for /8, /16, /24 and /32 part), Source UDP/TCP Port (One input), Destination IPv4 address (Four inputs, standing for /8, /16, /24 and /32 part), Destination TCP/UDP Port (One input) and Acknowledgement bit (One input). These twelve input parameters are to be processed for identification, by the router.

In the case of neural networks, the wild-card character is represented by two input patterns - the lower and upper bound of the corresponding parameter. In the case of addresses, the value is represented as 0 and 255

Table 1: Experiment 1 ACL Rules

Conditions						
Packet type	Source IPv4 address	Source TCP/UDP port	Destination IPv4 address	Destination TCP/UDP port	ACK bit	Action
TCP	*	*	202.185.*.*	*	Established	Permit
TCP	*	*	202.185.*.*	53	*	Permit
TCP	*	*	202.185.33.44	25	*	Permit
TCP	*	*	202.185.55.66	119	*	Permit
IP	*	*	*	*	*	Deny

* Not Available

Table 2: Experiment 2 ACL Rules

Source IP address	Destination IP address	Action
10.1.2.1	10.1.1.*	Deny
10.1.2.*	10.1.3.*	Deny
*	*	Permit

Table 3: Experiment 3 ACL Rules

Conditions						
Packet type	Source IP address	Source TCP/UDP port	Destination IP address	Destination TCP/UDP port	ACK bit	Action
TCP	*	*	10.1.1.2	www	*	Permit
UDP	0.0.0.*	*	10.1.1.1	*	*	Deny
IP	10.1.2.*	*	10.1.3.*	*	*	Deny
IP	*	*	*	*	*	Permit

Table 4: Experiment 4 ACL Rules

Conditions						
Packet type	Source IP address	Source TCP/UDP port	Destination IP address	Destination TCP/UDP port	ACK bit	Action
TCP	10.1.1.2	WWW	*	*	*	Permit
UDP	10.1.1.*	*	0.0.0.*	*	*	Deny
IP	10.1.3.*	*	10.1.2.*	*	*	Deny
IP	10.1.2.*	*	10.1.3.*	*	*	Deny
IP	10.1.1.130	*	10.1.3.2	*	*	Deny
IP	10.1.1.28	*	10.1.3.2	*	*	Deny
*	*	*	*	*	*	Permit

Table 5: Experiment 4 ACL Rules

Conditions						
Packet type	Source IPv4 address	Source TCP/UDP port	Destination IPv4 address	Services	ACK bit	Action
*	*	*	*	finger, bootp, udp-525, ident, login	*	Deny
*	202.185.128.*	*	202.185.11.*	http, smtp	*	Deny
*	*	*	202.185.*.*	smtp, imap, pop3	*	Deny
*	*	*	*	smtp, imap, pop3	*	Permit
*	202.185.131.*	*	*	*	*	Permit
*	202.185.128.*	*	202.185.130.1	http, tcp	*	Permit
*	*	*	202.185.130.3	http, tcp	*	Permit
*	202.185.130.2	*	202.185.128.*	netbeui	*	Permit
*	*	*	202.185.130.4	ftp	*	Permit
*	202.185.128.*	*	202.185.131.178 202.185.131.179	ftp	*	Permit
*	*	*	202.186.130.1	rpc, syslog tcp	*	Permit
*	202.185.130.4	*	202.185.130.5	*	*	Permit
*	202.185.128.*	*	202.185.130.5	telnet	*	Permit
*	202.185.131.170	*	202.185.129.120	*	*	Permit
*	202.185.128.* 202.185.130.1	*	202.185.131.*	http, ftp, tcp, telnet	*	Permit
*	*	*	202.185.130.5	tcp, dns, http, nntp	*	Permit
*	202.185.128.*	*	*	tcp, http	*	Permit
*	*	*	202.185.130.1- 202.185.130.5	http, tcp	*	Permit
*	202.185.129.*	*	*	dns, nntp, http, tcp, spool	*	Permit
*	202.185.128.*	*	202.185.131.*	snmp, icmp, echo	*	Permit

instead of giving all the values from 0 to 255. Only one output neuron is used to represent the Permit or Deny operation.

The network is trained to give the same output for the two extreme address values 0 and 255 and hence any value between 0 and 255 will give rise to the same output. This has been ascertained using the following mathematical justification.

In a neural network, the net input to a neuron is given by,

$$Net = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n \quad (1)$$

where, Net is the net input to the neuron

$x_1 \ldots x_n$ are the inputs

$w_1 \ldots w_n$ are the weights between the nodes

Let Net_1, Net_2 and Net_3 be the net inputs to the neuron obtained when the input is replaced by "a", "b" and "c" (in the place of x_1) respectively. The neuron is trained to have the same outputs when Net_1 and Net_2 are applied at its inputs. If "c" lies between "a" and "b" then for the same transfer function (say, sigmoidal) the output for an input of Net_3 to the neuron will have the same value of outputs as that of Net_1 and Net_2. The above statements are summarized in the following equations:

$$Net_1 = w_1a + w_2x_2 + w_3x_3 + \ldots + w_nx_n \quad (2)$$
$$Net_2 = w_1b + w_2x_2 + w_3x_3 + \ldots + w_nx_n \quad (3)$$
$$Net_3 = w_1c + w_2x_2 + w_3x_3 + \ldots + w_nx_n \quad (4)$$
$$\text{If } f(Net_1) = Out_1 \text{ and also if}$$
$$f(Net_2) = Out_1 \text{ and } a <= c <= b \text{ then } f(Net_3) = Out_1 \quad (5)$$

A typical neural network diagram for Back Propagation Algorithm is shown in Fig. 1.

The Rule Optimization Technique: Let us represent the values of the six parameters of the IPv4 header [Packet type, Source IPv4 address, Source TCP/UDP port, Destination IPv4 address, Destination TCP/UDP address, ACK bit] as X_{11}, X_{12}, X_{13}, X_{14}, X_{15} and X_{16}. The rules specified in the Access Control List can be denoted as R(i,j) where "i" indicates the rule number and "j" indicates the parameter number in the ACL.

Rule: Define $X_{ij} \in R(i,j)$ when $R(i,j) = *$, or when some part of the R(i,j) is a wild-card and the rest of the R(i,j) is equal to the corresponding part of X_{ij}.

For example, if X_{ij} = "202.185.33.44" and R(i,j) = "202.185.*.*", then $X_{ij} \in R(i,j)$. A Similarity function is defined as:

Similarity(i,j) = 1 if $X_{ij} \in R(i,j)$ = 0 Otherwise.

The condition C_j in the jth column of the ACL is defined as:

C_j = Similarity(1,j) $\wedge$ Similarity(2,j) $\wedge$ Similarity(3,j) $\wedge \ldots \wedge$ Similarity(m,j).

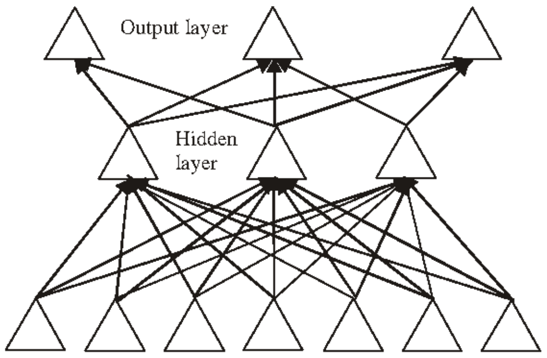


Fig. 1: Back Propagation Algorithm

Where, "m" is the number of rules in the ACL.

If C_j is equal to 1, then the corresponding parameter is further checked for the following conditions. Otherwise, optimized-sequential parsing is done.

If all the entries in the jth column are wildcards then that corresponding parameter is discarded and the next parameter will be checked. Else, if only a part of the parameter is matched, then the corresponding part can be removed.

Mathematically, the optimization procedure is represented as:

$\forall$i (Column[i] = *) || $\forall$i (Column[i] = Constant) Ignore the column

$\forall$i (Part-of-column[i] = *) || $\forall$i (Part-of-column[i] = Constant) Ignore that part of the column

$\forall$i $\exists$j (row[i] = row[j]) Ignore any one rule (row).

For example, the /8 part of all the rules of a parameter (Column of the parameter) is compared. If they match as per the Rule, then the fixed part is discarded. Then the process of matching is done for the /16 term and so on. All unmatched terms are considered for inputting to the filter.

After optimization is applied, only five parameters are used for training a neural network. They are Packet Type, Destination IPv4 address, Destination TCP/UDP Port and ACK bit. For Packet Type, 1 represents TCP, 2 represents UDP and 3 represents IP. For Destination IPv4 address, we have two inputs standing for /16 and /32 parts of the address. For Destination TCP/UDP Port, we have only one input. For ACK bit, Established is represented as 1, wildcard "*" as 0.

For processing the optimized rules, a Backpropagation Neural network is trained. The Neural Network has five input neurons, two hidden neurons and one output neuron. The selection of two hidden neurons is based on the following previous studies:

* A rule of thumb is for the size of this hidden layer to be somewhere between the input layer size and the output layer size [4].
* How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer [5].
* The number of hidden neurons should be less than the number of input neurons [6].
* The number of hidden neurons is calculated as [7].

Number of hidden neurons
= (Number of input + number of output)/2 (6)

Thus with five input neurons and one output neuron, we can have three hidden neurons as per [7]. We preferred to have two hidden neurons after testing the neural network with different number of hidden neurons. With hidden neurons = 1, the number of epochs was 41 but the data was trained well. With hidden neurons = 2, the number of epochs was 8 with actual output as 0.9955 instead of 1 [target output]. With hidden neurons = 3, the number of epochs was 17 with actual output as 0.9928 instead of 1. With hidden neurons = 4, the number of epochs was 12 with actual output as 1.0033 instead of 1. With hidden neurons = 5, the number of epochs was 10 with actual output as 0.9884 instead of 1. From the above, it is clear that the use of two hidden neurons perform better in terms of epochs and error in training.

EXPERIMENTAL DATA AND ANALYSIS

Here, we select a set of rules that are based on the extended access list format where only the source and destination addresses are used. Five sets of data were taken among which four are shown here.

Analysis of ACL Rules Using Neural Networks: The neural network system was designed with the access control rules discussed above. These rules were represented into the neural network system with back-propagation network algorithm. The network with inputs and action as the output was trained for all the five experimental access rules set presented in Tables 1-5. From the Tables 1-5 we can see that Experiments 2 and 4 alone have implicit rules (rules that posses no specific conditions but only indicate the action for those packets that don't satisfy the other access rules) at the end of the access lists. The number of rules in each access list varies from 3 to 21. These five experiment access lists are trained with eight different forms as input to the neural network system. The output of the system is either 0 or 1 indicating deny or permit respectively. According to the form of the input, the system architecture also changes as shown in Table 7. The following are the various forms of input for the neural system:

* ACL: The access list data were input as in the access list and replacing the wildcard by 0. Here, the problem was noted in the case of totally wildcard rules like that in Experiment 2 because all the inputs are zero there.
* ACLW: In order to cater for the range of the wildcard, we represented the wildcards with two sets of inputs as 0 and 255. Any access rule with wildcard is entered twice, with 0 replacing wildcards in one rule and 255 replacing wildcards in another.
* BACL: In order to make the neural network train faster, because decimal calculation might consume more time and the error rate might be high for decimal data, we converted the data into binary format. Decimal calculation also creates round-off errors due to insufficient precision. For the packet type, we had two bits representing it, TCP as 1, UDP as 2 and IP as 3. The source and destination addresses and ports were represented using eight bits. This made the number of inputs present in the neural system to increase. In Experiment 5, we did provide numbers for various services given there and input them as numbers of eight bits.
* BACLW: With BACL as the format, we added information for the wildcards using 0 and 255 representations in binary format. In most cases, the number of patterns was doubled by the use of 0 and 255 representations of the wildcards.
* ACL, ACLW, BACL and BACLW were done with optimized rules and they are named as OACL, OACLW, BOACL and BOACLW, respectively.

The network performance was noted for how long it takes for the neural network to train and whether the training process was completed successfully. The number of epochs (iterations) indicated whether the performance goal was met or not while the architectural representation of the network was shown in Table 7. Performance goal indicates the maximum allowed error between the output generated by the neural network system and the actual ideal output. In the case of architecture, we note the number of inputs present in each pattern. This number of inputs indicates the number of input neurons in the neural network system and the number of patterns indicates the number of output neurons needed for the neural network system.
From Table 7, we can infer that the neural network system has learned well in most cases and the performance goal was met not only in three cases. The neural network stops the training process when the performance goal is met (which means the error between the output of the neural network and the actual output is less than the allowed error) or when the change in weights is minimum and that this change doesn't make the neural network learn further. From

Table 6 : Comparison of all the Five Experiments

	Sequential [Comparison]		Opt. sequential [Comparison]		Neural network		Expert system [Comparison]	
	Min	Max	Min	Max	Addition	Multiplication	Min	Max
Ex1	12	60	5	25	7	11	1	4
Ex2	8	24	7	21	11	15	1	7
Ex3	12	48	9	27	12	30	1	10
Ex4	12	84	10	70	13	31	1	14
Ex5	12	290	9	210	12	28	1	18

Table 7: Performance Comparisons

		ACL	ACLW	BACL	BACLW	OACL	OACLW	BOACL	BOACLW
Ex 1	No. of Epochs	5	18	11	33	12	17	10	43
	Goal met?						No		
	Inputs/Patterns	12 / 5	12 / 10	83 / 5	83 / 10	5 / 5	5 / 10	27 / 5	27 / 10
Ex 2	No. of Epochs	3	6	6	8	3	5	5	14
	Goal met?								
	Inputs/Patterns	8 / 3	8 / 5	64 / 3	64 / 5	7 / 3	7 / 4	56 / 3	56 / 4
Ex 3	No. of Epochs	8	25	8	20	10	72	15	18
	Goal met?								
	Inputs/Patterns	12 / 4	12 / 8	84 / 4	84 / 8	9 / 4	9 / 8	56 / 4	56 / 8
Ex 4	No. of Epochs	26	24	14	23	14	38	21	23
	Goal met?								
	Inputs/Patterns	12 / 7	12 / 13	83 / 7	83 / 13	10 / 7	10 / 13	74 / 7	74 / 13
Ex 5	No. of Epochs	27	163	49	59	25	50	53	57
	Goal met?		No				No		
	Inputs/Patterns	21 / 12	21 / 42	83 / 21	83 / 42	9 / 21	9 / 42	72 / 21	72 / 42

Table 8: Impact on Number of Epochs

	ACL	ACLW	BACL	BACLW	OACL	OACLW	BOACL	BOACLW	Mean	Impact on experiments
Ex 1	5.0	18.0	11.0	33.0	12.0	17.0	10.0	43.0	18.625	-7.40
Ex 2	3.0	6.0	6.0	8.0	3.0	5.0	5.0	14.0	6.25	-19.775
Ex 3	8.0	25.0	8.0	20.0	10.0	72.0	15.0	18.0	22.00	-4.025
Ex 4	26.0	24.0	14.0	23.0	14.0	38.0	21.0	23.0	22.875	-3.15
Ex 5	27.0	163.0	49.0	59.0	25.0	50.0	53.0	57.0	60.375	34.35
Mean	13.8	47.2	17.6	28.6	12.8	36.4	20.8	31.0	26.025	
Impact on methods	-12.225	21.175	-8.425	2.575	-13.225	10.375	-5.225	4.975		

Table 9: Experimental Errors in the Number of Epochs

	ACL	ACLW	BACL	BACLW	OACL	OACLW	BOACL	BOACLW
Ex 1	-1.40	-21.8	0.80	11.80	6.60	-12.00	-3.40	19.40
Ex 2	8.975	-21.425	8.175	-0.825	9.975	-11.625	3.975	2.775
Ex 3	-1.775	-18.175	-5.575	-4.575	1.225	39.625	-1.775	-8.975
Ex 4	15.35	-20.05	-0.45	-2.45	4.35	4.75	3.35	-4.85
Ex 5	-21.15	81.45	-2.95	-3.95	-22.15	-20.75	-2.15	-8.35

this table, we can also infer that the binary form of neural network requires additional iterations to train due to the increase in number of inputs. The number of rules present in each experiment determines the number of patterns. Even though converting to binary increases the number of input nodes and with it the complexity of the network architecture, the number of epochs doesn't seem to be affected much due to the fact that the binary operations are performed faster than the decimal operations.

The results of the analysis (Table 8) are interpreted as follows : An average experiment using an average method of access control requires 26f.025 epochs. The number of epochs needed for ACL method is 12. 225 epochs less than the average. The number of epochs required is less than average for BACL, OACL and BOACL methods. But, the number of epochs required is more than average for other methods.

The selection of the experiment also impacts the performance of the access control lists. An average experiment requires 26.025 epochs. All except experiment 5 require fewer epochs than average. This result might be due to the fact that experiment 5 is the access control list that currently rules on a live university network and all others are test phase access lists.

We can find out the experimental errors by finding the difference between the estimated response and the measured response, as shown in Table 9.

Sum of Squared Errors (SSE) = 13919.95

Sum of Squared Measured Responses [y_{ij}] SSY = $\sum_{ij} y_{ij}^2$ = 58277

Sum of squared overall mean (SS0) =
$ab\mu^2$ = 8*5*(26.025)^2 = 27092.025

Sum of Square of α_j (SSA) = $b\sum_j \alpha_j^2$ =
5*[(-12.225)^2 + (21.175)^2 + (-8.425)^2 + (2.575)^2 + (-13.225)^2 + (10.375)^2 + (-5.225)^2 + (4.975)^2]
= 5050.175

Sum of Squares of β_i (SSB) = $a\sum_i \beta_i^2$ = 8*[(-7.4)^2 + (-19.775)^2 + (-4.025)^2 + (-3.15)^2 + (34.35)^2] = 13214.85

Sum of Squares Total (SST) = SSY – SS0 = 58277 - 27092.025
= 31184.975

Sum of Squared Errors (SSE) = SST – SSA – SSB = 31184.975 - 5050.175 – 13214.85 = 12919.95

The percentage of variation explained by the various ACL methods is

100 * SSA/SST = 100 * [5050.175/31184.975] = 16.19%

The percentage of variation explained by the various ACL rules is

100 * SSB/SST = 100 * [13214.85/31184.975] = 42.38%

The unexplained variation is

100 * SSE/SST = 100 * [12919.95/31184.975] = 41.43%

Looking at these percentages, it is found that the choice of ACL rules is an important parameter and it plays a vital role than the methods themselves. But still there is considerable impact by the rules selection.

Neural Network Test Data Analysis: After training the neural network in various different forms as mentioned earlier, we tested the system with three data sets for each experiment. These data sets where selected to see the consistency of the trained neural network system with the actual results the access control rules provide. The test data set for all the experiments is shown in Tables 1-14.
The neural network system with all the above data set was tested for security aspects. The actual output expected for these data sets is shown in Table 15 as Idle Value. The output from the various neural network systems is shown in Table 15.

From Table 15, we can infer the following:

* The neural network that is of ACL rules alone has not trained the system and so none of the experiments gave accurate result for all the three test data sets.
* Experiment 2 was not properly learnt due to the following rules:
* Presence of very few numbers of rules in the ACL rules list and one of them is an implicit permit. Being implicit makes the neural network difficult to learn. Representing the implicit as 0 makes the neural network learn only for 0 or closer values and representing the implicit as 0 and 255 makes the neural network confuse the other set of rules. The impact of sequential set of rules also has an impact on the neural network training process.
* Referring to the number of epochs from Table 7, the value is less compared to other experiments. This doesn't mean that the neural network has trained faster and better. As already known, fast learning doesn't mean that the neural network has learnt better.
* Optimizing the ACL rules and using them to train the neural network leads to some improvement. Experiment 3 and 5 are trained using OACL. But the impact by representing 0 and 255 as wildcards to OACL is not much. The outcome of the system is the same with OACL and OACLW. So, there is no need of replacing wildcard with 0 and 255 that causes the increase in the number of patterns and make the architecture of the neural network complex.
* The use of binary input for the OACL makes some difference in the output values but the permit/deny decisions remains the same.
* The case of OACLW and ACLW for Experiment 5 is ambiguous because the system couldn't meet the performance goal. This is due to the fact that the neural network would have fallen into the local minima and so the weights cannot be further adjusted to train the network better.
* The relationship between the number of epochs and training is normally that with the increase in the number of epochs, the system trains better. But care must be taken that the above applies only when the performance goal is met. The case of OACLW for Experiment 3 is an example.

From Table 15 it is clear that none of the neural network model can identify the entire test set for all the experiments properly. Only the BOACLW could learn three of the experiments properly. Experiment 2 is not taken into consideration as it has less number of rules and one of them is of implicit type. For Experiment 1 (in the case of BOACLW) with little problem of identifying the deny option, the network has learnt better than the ACL rules.

Table 10: Test Data for Experiment 1

Conditions						
Packet type	Source IP address	Source TCP/UDP port	Destination IP address	Destination TCP/UDP port	ACK bit	Action
IP	11.12.13.14	15	21.22.33.44	119	1	Deny
TCP	11.12.13.14	15	202.185.25.100	53	0	Permit
TCP	11.12.13.14	15	21.22.100.100	100	1	Permit

Table 11: Test Data for Experiment 2

Source IP address	Destination IP address	Action
10.1.3.4	10.10.10.7	Permit
10.1.2.5	10.1.1.7	Permit
10.1.2.1	10.1.1.7	Deny

Table 12: Test Data for Experiment 3

Conditions						
Packet type	Source IP address	Source TCP/UDP port	Destination IP address	Destination TCP/UDP port	ACK bit	Acion
TCP	10.10.10.10	10	10.1.1.2	www	0	Permit
TCP	10.10.10.10	10	10.1.1.2	www	1	Permit
IP	10.1.2.100	100	10.1.3.100	100	0	Deny

Table 13: Test Data for Experiment 4

Conditions						
Packet type	Source IP address	Source TCP/UDP port	Destination IP address	Destination TCP/UDP port	ACK bit	Action
TCP	10.1.1.2	80	10.10.10.10	100	1	Permit
IP	10.1.3.25	100	10.1.2.25	100	0	Deny
IP	10.1.1.28	100	10.1.3.2	100	1	Deny

Table 14: Test Data for Experiment 5

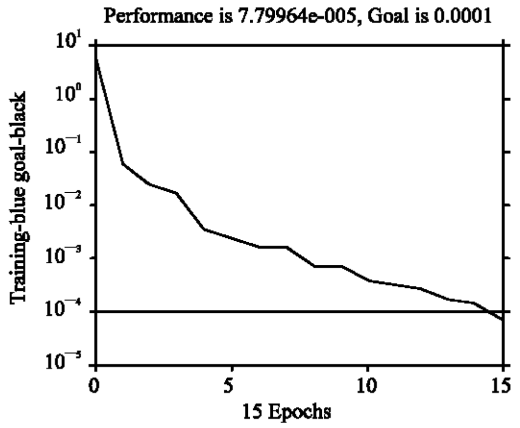
Conditions						
Packet type	Source IP address	Source TCP/UDP port	Destination IP address	Services	ACK bit	Acion
TCP	202.185.128.100	100	202.185.131.100	ftp	0	Permit
TCP	202.185.128.50	50	202.185.130.5	telnet	1	Permit
TCP	202.185.128.4	50	202.185.130.5	http	0	Permit

Table 15: Security Comparison Among Neural Network Packet Filtering Systems

		Idle value	ACL	ACLW	BACL	BACLW	OACL	OACLW	BOACL	BOACLW
Ex 1	Test Data 1	0	0.8823	0.3403	0.4189	0.8356	1.0077	1.0000	0.5381	0.7244
	Test Data 2	1	0.9942	0.9984	1.1913	0.9628	0.1107	0.8021	1.0986	1.1419
	Test Data 3	1	0.8823	1.9925	0.7438	0.5051	1.0077	0.8000	1.1508	1.1945
Ex 2	Test Data 1	1	-0.0427	-0.0215	0.0577	0.0817	0.4743	0.5788	1.6736e-005	0.1026
	Test Data 2	1	-0.0536	-0.0274	-0.0299	0.0088	-4.9951e-004	0.0467	1.8616e-005	0.0018
	Test Data 3	0	-0.0522	-4.8718e-004	0.0292	0.0061	3.4158e-005	0.0039	1.9675e-005	-5.4934e-004
Ex 3	Test Data 1	1	1.0012	0.2060	1.1292	0.4495	1.0007	1.0128	0.4626	0.8048
	Test Data 2	1	1.0012	0.1965	1.2884	0.5794	1.0007	1.0128	0.4626	0.8048
	Test Data 3	0	1.3233	-0.0071	0.0434	-0.0632	0.2597	0.0031	-0.0119	0.3286
Ex 4	Test Data 1	1	0.2538	0.0925	0.4244	0.7866	0.9913	0.9870	-0.0412	1.1817
	Test Data 2	0	0.2700	-0.0037	0.3800	0.0533	0.9913	0.6910	0.0328	-0.3673
	Test Data 3	0	1.0119	-0.0057	-0.1730	-0.2159	-0.2101	1.1868	0.0044	-0.2688
Ex 5	Test Data 1	1	0.3251	2.6991	1.1904	0.4980	0.8116	1.0363	0.9806	0.7618
	Test Data 2	1	0.9801	1.9310	1.2490	1.3889	1.0013	0.8803	1.0240	0.9234
	Test Data 3	1	1.0024	0.9749	0.8188	0.3861	1.0013	1.0253	1.0066	0.9975

Table 16: Input for the Neural Network System

CPU use	PKTS	PAGE	SWAP	INTR	DISK	CNTXT	LOAD	COLLS	ERROR	Status	Target Value
20	539	6	0	564	26	724	157	0	0	OK	1.0121
23	543	0	0	666	18	728	157	0	0	OK	1.0075
27	708	18	0	663	61	1052	155	0	0	OK	0.9931
29	807	0	0	555	0	1869	53	0	0	OK	1.0187
39	1005	0	0	856	11	1301	159	0	0	OK	0.9947
39	1262	0	0	1011	180	3497	163	0	0	OK	0.9955
40	1019	4	0	951	25	1762	156	0	0	OK	0.9984
42	1075	0	0	892	17	1338	154	0	0	OK	0.9984
59	626	3	0	836	196	1241	162	0	0	OK	0.9930



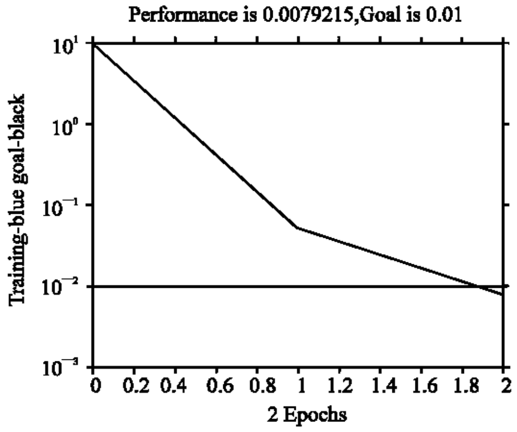


Fig. 2: Training Stages for the Maximum Error of 0.0001 and 0.01

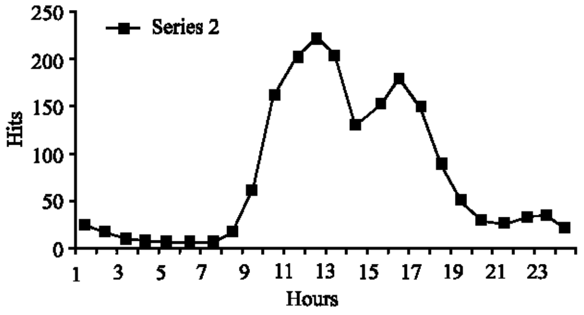


Fig. 3: The Network Usage Graph in Hourly Basis

Niels [8] feels that access control list can extend the traditional access model to provide finer-grained controls but can not prevent untrusted applications from causing damage. As the neural network system that uses access control list alone is not complete, we include a next level of security check using the local data obtained from the network and also the hourly analysis of the network usage. The primary level of neural network training were done based on access control rules. Analysis shows that there are not sufficient enough to take care of the security of the network. In order to improve the security, we did consider developing a second level of security check.

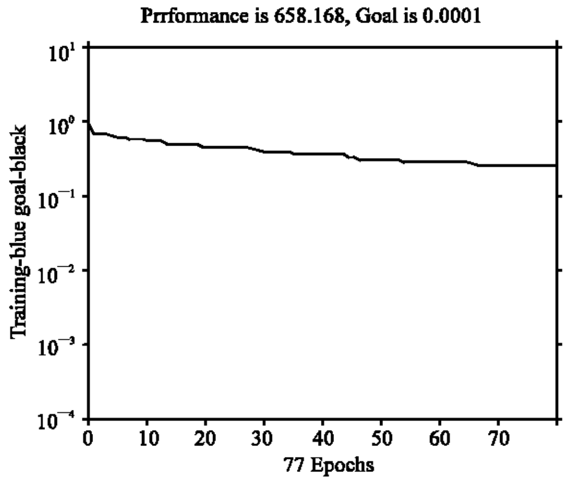


Fig. 4: The Training Process of Neural Network

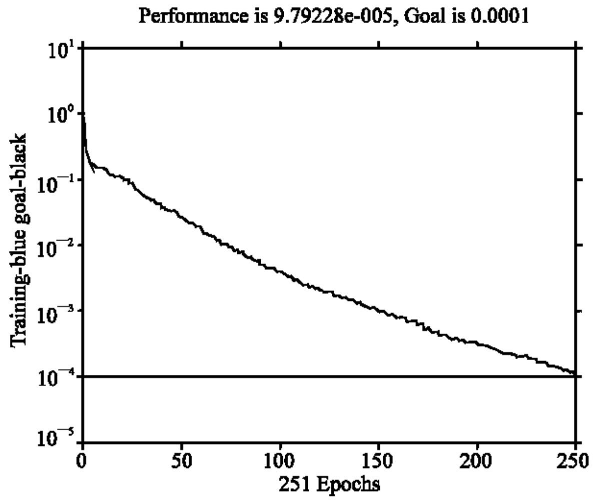


Fig. 5: The Training Process for Binary Data

This security check using neural networks was done using the local data obtained from the network along with the hourly analysis of the network usage.

Local Data Oriented Neural Network Analysis: Matching of the past data with the recent data is difficult and a cumbersome task if the past data is huge. False alarms can result from invalid assumptions about the distribution of the audit data made by the statistical algorithms [9]. Missing some data can end up in a dilemma for discriminating between an intruder and a normal user. So, the rule-based approach was used to fill the gaps present in the statistical algorithms. Writing such a rule-based system is a knowledge-engineering problem and the resulting “expert system” will be no better than the knowledge and the reasoning principles it incorporates. Maintaining a large rule-base is highly complex.

The inputs provided for this neural network were in binary form. Various data sets were collected and converted into binary and were used to train both the networks. The number of input nodes was 69. These

Table 17: Input Data During Testing Stage Along with its Group

CPU use	PKTS	PAGE	SWAP	INTR	DISK	CNTXT	LOAD	COLLS	ERROR	Status	Value
10	300	3	0	511	17	612	153	0	0	Abnormal	0.7555
25	622	9	0	671	40	1043	160	0	0	Ambiguous	1.1388
35	962	0	0	1012	35	1967	161	0	0	OK	1.0976
50	714	2	0	947	60	1276	160	0	0	Abnormal	0.6451
70	564	1	0	863	45	1112	161	0	0	Ambiguous	1.1638
10	962	3	0	511	60	1967	253	0	0	Abnormal	0.7925
25	714	9	0	1012	45	1276	160	0	0	OK	1.0874
35	300	0	0	671	35	1967	254	0	0	OK	1.0743
50	622	2	0	947	40	612	160	1	1	OK	0.9335
70	564	1	0	863	35	1043	241	0	0	Ambiguous	1.1541

Table 18: Number of Hits on Hourly Basis

Hour	Hits
00	21.68
01	16.85
02	9.27
03	8.55
04	6.64
05	7.00
06	5.58
07	13.00
08	61.87
09	158.33
10	197.07
11	218.74
12	202.90
13	131.00
14	151.69
15	174.05
16	149.41
17	89.37
18	51.14
19	26.97
20	26.46
21	33.51
22	35.95
23	23.35

input nodes represent the usage of CPU [7 bits], Number of packets that were used for transmission [11 bits], number of pages accessed [5 bits], Number of pages swapped [1 bit], Number of interrupts created [11 bits], disk usage [8 bits], context switching that occurred [16 bits], total load [8 bits], number of collisions [1 bit], number of error occurred [1 bit]. The training rate was fixed at 0.1. The number of epochs for each training stage was fixed at 500 and the following were inferred.

A Back Propagation Neural Network with 69 binary input neurons, two hidden neurons and one output neuron was trained. The number of binary input neurons depends on the input data we use to train the network. We used the concept of training the neural network system with the help of user data and identification of misuses of the resources, with the deflection in the usage of the system. Table 16 shows the part of the input used for training the neural network system, whose output is trained for the normal user, indicating that the system is given a data set that represents a proper user. Table 16 also shows the values that were attained during the testing stage, when the trained data was given as input. It is clear from this table that we have trained the system only for a normal user. So, any deviation from this type of data is considered to be abnormal.

After the network was trained, some normal as well as abnormal user data were provided to the system as shown in Table 17. The system could recognize some of the inputs clearly, but at the same time the cases whose value [target value and the measured value] have a difference of more than 0.1 are considered to be decided based on the regularity of operation of the network. If the network has some fluctuations normally, then the values with more than 0.1 are considered as normal data. If the network usage is almost stable, then the difference of 0.1 or more is considered as abnormal. The training of the Back Propagation based Neural Network was smooth. For a maximum error of 0.0001, it took only 15 epochs. This smooth training is visible from Figure 2. It is also clear from those Fig. 2 that to have a smooth and normal training step instead of a step-wise training, we should increase the maximum error. The system takes more epochs to reach the maximum error rate of 0.0001 than 0.01.

The variation in the input is due to the fact that the usage of the system resources varies from person to person and also varies according to the timing of the data collection. The major problem is that if a subnet is used by many people who are of different style of jobs, then the variations will be high. These variations will cause some problems for the network to get trained. To avoid this problem, we try to get neural networks trained for each individual user separately. As the trained neural network can test the data faster, it does not take much time if we need to test the data of few members in a single subnet at a specified time. For a single user, if the variations are high, we train the system with the respective time group of data acquisition. As per the columns that are having zero data, the indication is that the usage is not heavy and so there is no collision, error and swapping.

Hourly Hits Oriented Neural Network Analysis: The office is used to have peak working time at around 10 to 12 noon. In order to visualize the usage of the server at various hours, we need to look at how many times a server has been accessed on a day basis. The usage of the server varies based on the timing of the day and the day of the week.

With the Table 18 that we obtained from our university network, we can notice that the accessing of the system varies on the day in different levels. The usage of the system is higher during office hours and low during the night hours of the day. Lunch time contributes to little degradation in the usage of the system and the number of hits reduces at that point of time. The pattern of this access is shown in Fig. 3.

We have designed the neural network system to learn the pattern of the network access and then indicate whether there is any possible abnormal activity. We did implement a Backpropagation based neural network system with two inputs and one output. The inputs represent the hour of the day and the number of hits and the output represents the number of hits that is expected in the following hour. A neural network system with above information was implemented with decimal data, but the system cannot be trained. The neural network system stops because the difference in the output observed between the current and previous state is much less and not the required change needed to move towards the totally trained state. Being decimal in nature, the difference in the number of inputs between various hours is not much. But when converted into binary there are a number of bits which vary and the difference is present in more than one input.

From Figure 4, it is found that the neural network cannot learn. We can also refer to Fig. 4 and realize that the error [which is the vertical axis] is very high. The system stops with the condition that the decrease in the error is very minimum as shown below.

```
TRAINCGB-srchcha, Epoch 0/500, MSE 10553.5/0.0001, Gradient 1200.93/1e-006
TRAINCGB-srchcha, Epoch 25/500, MSE 2025.6/0.0001, Gradient 108.738/1e-006
TRAINCGB-srchcha, Epoch 50/500, MSE 867.637/0.0001, Gradient 157.976/1e-006
TRAINCGB-srchcha, Epoch 75/500, MSE 658.194/0.0001, Gradient 1.16254/1e-006
TRAINCGB-srchcha, Epoch 77/500, MSE 658.168/0.0001, Gradient 2.40323/1e-006
TRAINCGB, Minimum step size reached, performance goal was not met.
```

We then converted the decimal values into binary format and implement the back propagation network with binary values. The number of inputs were 13 and the number of outputs were 8. The hours of the day [0 to 23] need five binary bits and the number of hits needs eight binary bits. As the output is the expected number of hits in the next hour the number of bits needed for output is also eight. This network was trained as shown in Figure 5.

As shown in Fig. 5, the network can train well when we used the binary data but the number of epochs needed to train is a large value. But the training is smooth. In order to speedup the training process, we can consider doing the process in parallel over the network by using PVM daemons. We can have more daemons running during the peak hours to cater huge network utilization during that specific time. Thus the process of doing actions in parallel might help the neural network oriented parallel filtering process to act faster.

CONCLUSION

In this study, we have implemented various input representations for IP packet filtering. Different neural network architectures based on back propagation algorithm were tested and trained with eight input sets. Those neural network architectures were analysed for the performance of the training process and the correctness of their learning. Analysis indicates that neural network could not learn all the experiments well and the presence of implicit rules for denial/permission plays a pivotal role in the training of the neural network. In order to improve further on the security aspects, we designed neural network that could be trained with local user data and the hourly hits on servers. After the training, we are able to conclude that the neural network learns better with binary input data but at the same time this binary input data increases the input nodes and thereby the complexity of the network. Improving the security aspects of the neural network with various sets of data along with the binary form affected the performance of the neural network system. We conclude that usage of neural network for packet filtering is questionable because the inclusion of extra security features like local or hourly hits to take care of the security lapse in neural network system causes the performance gain to be affected. As future work, we could implement Intrusion Detection System [IDS] along with neural network system so that we can still attain the efficiency of the neural network system but get rid off the security lapse.

REFERENCES

1. Miei, T., M. Maruyama, T. Ogura and N. Takahashi, 1997. Parallelization of IP-Packet filter rules. Proceedings of Third International Conference on Algorithms and Architectures for Parallel Processing, 91: 381-388.
2. Murthy, U., O. Bukhres, W. Winn and E. Vanderdez, 1998. Firewalls for Security in Wireless Networks. IEEE Proceedings of 31st Hawaii International Conference on System Sciences, 7: 672-680.
3. Gittleson, H., R. Sharp and B. Cheswick, 1998. Red-hot firewalls. America's Network, 102: 48-52.
4. Blum, A., 1992, Neural Networks in C++, NY: Wiley.
5. Berry, M.J.A. and G. Linoff, 1997. Data Mining Techniques. NY: John Wiley and Sons.
6. Lawrence, S., C.L. Giles and A.C. Tsoi, 1997. Lessons in neural network training: overfitting may be harder than expected. Proceedings of the fourteenth national conference on artificial intelligence, AAAI-97, AAAI Press, Menlo Park, California, pp: 40-545.
7. http://hsb.baylor.edu/ramsower/ais.ac.97/paper/kwon.htm
8. Niels Provos, 2002. Improving Host Security with System Call Policies. (CITI Technical Report 02-3). Center for Information Technology Integration, University of Michigan.
9. Teresa, F.L., A. Tamaru, F. Gilham, R. Jagannathan, P.G. Neumann and C. Jalali, 1990. IDES: A progress report. Proceedings of the sixth Annual Computer Security Applications Conference, 90: 273-285.