NETWORK PERFORMANCE TUNING USING REINFORCEMENT LEARNING

G. Prem Kumar and P. Venkataram
Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore - 560012 INDIA
E-Mail : {prem,pallapa}@ece.iisc.ernet.in

ABSTRACT

Performance degradation in communication networks are caused by a set of faults, called *soft failures*, owing to which the network resources like bandwidth can not be utilized to the expected level. An automated solution to the performance management problem involves identifying these soft failures followed by the required change in few of the performance parameters to overcome the performance degradation. The identification of soft-failures can be done using any of the fault-diagnosis model [9], and is not discussed here. In this paper, the emphasis is on providing a viable solution to the network performance tuning. Since the exploration of the network performance tuning search space to bring out the required control is not well known in advance, a reinforcement control model is developed to tune the network performance. Ethernet performance management is taken up as a case study. The results obtained by the proposed approach demonstrate its effectiveness in solving the network performance management problem [11].

INTRODUCTION

Performance management is a complex part of present day network management [6]. Any abnormal behavior of the network in the absence of hard-failure is supposed to be handled by the network performance manager. Network performance degradation is viewed to be caused by a set of faults, called *soft-failures*. Solution to the network performance management problem involves two steps. Firstly, identify the soft-failures that cause the performance degradation. Secondly, tune the network parameters to bring its performance back to normalcy.

In another work [9], we proposed to use realistic abductive reasoning model (Realistic_ARM) to identify the network soft-failures. Each of the soft-failures is expected to have a remedy in the form of tuning up of network performance parameters. Performance tuning is done by using a neural network model that applies necessary correction to some of the parameters of the degraded communication network to bring it back to normalcy. A reinforcement learning mechanism, called *stochastic real valued algorithm* [12], is used for on-line tuning of the network. Based on the improvement in the network performance, the learning process takes place by adapting to the behaviour of the communication network.

In this paper, the soft-failure identification is not discussed. Any of the fault-diagnosis models which arrive with a set of explanations for a given set of abnormalities, each with some assigned probability is assumed to be the input to this work. A model for network fault diagnosis using realistic abductive reasoning model that also assigns probability to each of the explanations is discussed in [10]. A model to identify the soft-failures in Ethernet using Realistic Abductive Reasoning is presented below. Reinforcement learning mechanism for performance tuning and the results of the performance tuning for Ethernet follow it.

SOFT-FAILURE IDENTIFICATION IN ETHERNET

We consider a restricted Ethernet model to illustrate the soft-failure identification. For details on the Ethernet operation, refer [2][7].

We consider an Ethernet performance management model with the following assumptions.

- The information that needs to be *monitored* for the purpose of performance tuning is collected from the nodes and the channel. And, that information which is beyond the normal values (both above and below the normal limits) is reported as symptom(s).
- Some monitoring information like *load is normal* and *collisions are within the range* are included

No.	Fault	Remedy
1.	(F1)	(R1) : Faulty Ethernet card, report to the network manager
2.	(F2)	(R2) : Request the network manager to initiate fault diagnostic measures
3.	(F3)	(R3) : Ensure many packets are not above the specified size
4.	(F4)	(R4) : Report to the network manager
5.	(F5)	(R5) : Allocate more primary memory to the required nodes
6.	(F6)	(R6) : Selectively control the broadcast packets
7.	(F7)	(R7) : Report to the network manager along with the specified tap
8.	(F8)	(R8) : Ensure many packets are not below the specified size

Table 1. Suggested remedies for the soft-failures

to support the diagnostic process for minimizing the number of explanations.

- There may be some missing information and the entire information may not be available at the time of performance tuning.

PM Knowledge Model

The Ethernet performance management knowledge base [2][3][4][5][7] is constructed as a hyper-bipartite network (see Figure 1). This maps the Ethernet performance management knowledge onto a model suitable for the Realistic_ARM.

The Table 1 describes the suggested remedies for the soft-failures of the Ethernet model.

The Results

To identify the soft-failures, the algorithm Realistic_ARM is run by setting "the prespecified number of symptoms required to support an observed symptom before concluding a fault" to 1. The algorithm is run for various sets of symptoms (from layer 1 of Figure 1) and some of the results are given in Table 2. The probabilities are assigned to each of the links (not shown here), which have to be provided by the expert of the particular network.

From Table 2, it may be observed that the covers generated by the proposed model contain appropriate

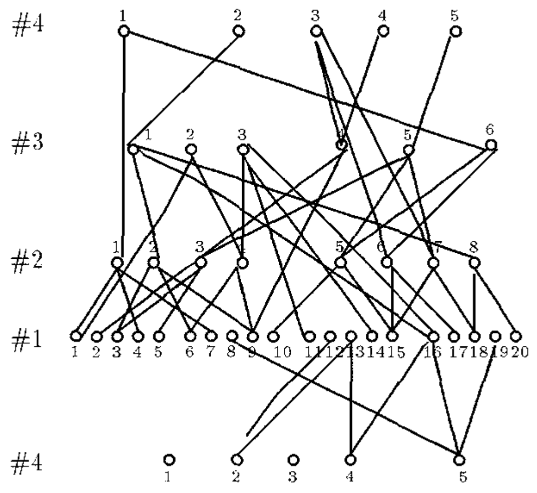

Fig. 1. Ethernet performance management knowledge model.

Legend:

Layer #1 :

1. Packet loss below normal; 2. Packet loss normal; 3. Packet loss above normal; 4. Load below normal; 5. Load normal; 6. Load above normal; 7. Collisions below normal; 8. Collisions normal; 9. Collisions above normal; 10.Large packets below normal; 11.Large packets normal; 12.Large packets above normal; 13.Small packets below normal; 14.Small packets normal; 15.Small packets above normal; 16.Broadcast packets normal; 17.Broadcast packets above normal; 18.Packet loss on spine above normal; 19.Load on spine normal; 20.Load on spine above normal

Layer #2 :

1. Light traffic; 2. Heavy traffic; 3. Buffers are insufficient; 4. Users are too many; 5. Preambles are too many; 6. Broadcast packets are too many; 7. Spine flooded with too many small packets; 8. Heavy traffic on spine

Layer #3 :

1. (F1) Babbling node; 2. (F2) Hardware problem; 3. (F3) Jabbering node; 4. Too many retransmissions; 5. Under utilization of channel as many small packets are in use; 6. Attempt for too many broadcasts

Layer #4 :

1. (F4) Bridge down; 2. (F5) Network paging; 3. (F6) Broadcast storm; 4. (F7) Bad tap; 5. (F8) Runt storm

No.	Symptoms	Remedy
1.	3,6,12,18,20	{ R5 }
2.	1,4,10,15,17	{ R4 }
3.	3,9,18,20	{ R1 }
4.	10,15,16,18	{ R8 }

Table 2. Sample results for Ethernet performance model.

explanation for any given symptoms without much of extra guess.

Consider a case given in S.No. 4 of Table 2. The observed symptoms in layer #1 are: Symptom 10 : number of large packets below normal; Symptom 15 : small packets above normal; Symptom 16 : number of broadcast packets are normal; Symptom 18 : packet loss on spine above normal.

The soft-failure concluded is "Runt storm" (with some plausibility) and the remedy is to ensure that too many small packets are not injected into the network.

NETWORK PERFORMANCE TUNING

Network performance management cannot be complete without a control mechanism that tunes the network to overcome the performance degradation.

When there is a degradation in network performance, we first identify the soft-failures with the plausibility of their existence and then take up the performance tuning. A *reinforcement learning* (RL) paradigm is employed to accomplish performance tuning. We use a RL model, with two modules, similar to that discussed in [8]. The first module, called the "actor", learns the correct action for a given agent and the second module, called the "critic", learns the maximum reinforcement expected from the environment for a given input.

Reinforcement Learning

Reinforcement learning problems are usually modeled as an *agent* interacting with an *environment* (see Figure 2). The agent receives an input from the environment and outputs a suitable action. The environment accepts the action from the agent as input and outputs a scalar evaluation (reinforcement) of the action as well as the next input to the agent. The agent's task is to learn how to react to a given input so as to maximize the reinforcement it receives from the environment.

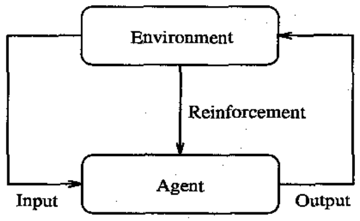

Fig. 2. The reinforcement learning mechanism

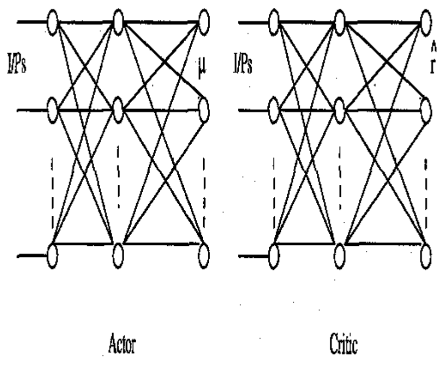

Fig. 3. A connectionist model for network PM.

In supervised learning, the agent is provided with the desired output from which it can form a gradient to follow so as to minimize the error in the output. In RL, such an explicit gradient information is absent. Hence the agent has to explore the output space to find the right direction.

A reinforcement learning is suitable to use for network performance tuning because, the desired output to bring the performance back to normalcy is not known well in advance.

The Neural Network Model

A connectionist model, as described in [12], is used to realize the reinforcement learning. This model uses two neural networks, each of them based on the back-propagation learning algorithm (see Figure 3.)

The neural network has an input layer that is designed to accept the significant events of environment as inputs.

The output of NN is the value on which the correction to the communication network parameter depends.

The NN also has one or more hidden layers, depending on the complexity of the problem. The number of neurons in the hidden layer(s) also depends on the estimate of the number of exemplars provided for training [1].

Each layer of the NN consists of one or more neurons. The output of one layer is fed as the input of the next. Associated with each of these connections is an adaptive weight. The output of a neuron in a hidden layer or output layer is calculated as an activation function of the weighted sum of the input to the neuron with an optional bias term, which is also, in principle adaptive.

For every pattern (or exemplar) presented to the neural network for training, two passes are used. The forward pass is used to evaluate the output of the neural network for the given input with the existing weights. In the reverse pass, the difference in the neural network output with the desired output is compared and fed back to the neural network as an error to change the weights of the neural network.

For the input layer neurons, the user input is applied and at the other neurons the net-input is calculated by using the following :

$$\sum_j (w_{ji} * input_j)$$

where w_{ij} is the weight of the link connecting the neuron i in the current layer and the neuron j in the preceding layer, and $input_j$ is the output of the neuron j fed to neuron i. The input to bias neuron in the preceding layer is set to -1 and the weight is adjustable. Essentially, the weight on the bias link in the previous layer indicates the threshold of each neuron in the current layer.

The output of each neuron is determined by applying a transfer function $f(.)$ to the net-input to the neuron. The commonly used functions are sigmoidal functions ($e.g.$, $(1+e^{-x})^{-1}$; and hyperbolic tangent, $tanhx$).

We use the function

$$f(x) = (1 + e^{-x})^{-1}. \tag{1}$$

In the reverse pass, $i.e.$, for the particular neuron i in output layer L, suppose that the output of the neuron is y_i^L and the desired output is d_i^L. The back-propagated error is

$$\delta_i^L = f'(h_i^L)[d_i^L - y_i^L]. \tag{2}$$

where h_i^L represents the net-input to the i^{th} unit in the L^{th} layer and $f'(.)$ is the derivative of $f(.)$.

For layers l = 1, . . ., (L-1), and for each neuron i in layer l the error is computed using :

$$\delta_i^l = f'(h_i^l)\sum_j (w_{ij}^{l+1} * \delta_j^{l+1}) \tag{3}$$

The adaptation of the weights of the link connecting neurons i and j is done using :

$$w_{ij}^l = w_{ij}^l + K * \delta_i^l * y_j^{l-1} \tag{4}$$

where K is the learning constant that depends on the rate at which the neural network is expected to converge.

This neural network algorithm essentially minimizes the mean square error between the neural network output and the desired output using gradient descent approach.

RL for Performance Tuning

The realistic abductive reasoning model evaluates the performance degradation with a set of soft-failures and respective plausibilities. These values are fed to the two modules of neural network. One of the neural networks is the "actor" and the other is the "critic". Let the output of the "actor" be μ, and that of the "critic" be $\hat{r}$ which is the goodness of μ. The correction z is applied to the communication network which is an instance of a Gaussian distribution with mean μ and variance σ where σ is a non-increasing function of $\hat{r}$. The variance term is used to control exploration of the output space by having a small value when $\hat{r}$ is close to the maximum value and otherwise a large value. Once again, the realistic abductive reasoning model evaluates the communication network to yield a new set of soft-failures and respective percentage degradation in performance. And, based on that, the reinforcement is given to both the "actor" and the "critic". The error used to adapt the weights of both networks is a function of the values of μ, r, and r. This process of soft-failure identification and performance tuning repeats till the network performance is brought back to normalcy.

The following are the equations for "actor" and "critic" neural networks that are incorporated into the connectionist model.

"Actor" neural network :

input x : the degradation in performance; plausibility of the performance parameter being controlled by the neural network; current setting of the parameter; other parameters of significance.

output : μ.

r = 1 if the performance improves and -1 if it degrades further.

backpropagated error : (r - $\hat{r}$) * (z - μ)/σ.

where z = f(μ, σ), where $f(.)$ is a Gaussian with mean μ and variance σ.

```
"Critic" neural network :
    input : x.
    output : $\hat{r}$.
    $\sigma = \max(0, (1 - \hat{r}))$.
    r = 1 if the performance improves and -1 if it degrades further.
    backpropagated error = $(r - \hat{r})$.
Algorithm Performance Tuning
Begin
    1. Choose a neural network with $1, \ldots, L$ layers.
    2. Set the input of the bias neurons to -1.0
    3. Initialize $w_{ij} = \delta \; \forall i, j$, a random value in the range (0,1).
    4. feed the input parameters
    5. Propagate the signal forward through the neural network till the output layer, evaluating output of each of the neurons $i$,
       $f(\sum_j (w_{ji} * input_j))$
       where $input_j$ is the output from neuron $j$ in the preceding layer, fed to neuron $i$.
    6. For each of the neuron $i$ in the output layer (layer L), compute the error $\delta_i^L$ using the equations given in the respective $\mu$ and $\hat{r}$ networks.
    7. For l = 1, ..., (L-1), compute the error
       $\delta_i^l = f'(h_i^l) \sum_j (w_{ij}^{l+1} * \delta_j^{l+1})$
    8. Update the weights using :
       $w_{ij}^l = w_{ij}^l + K * \delta_i^l * y_j^{l-1}$
       where $K$ is the learning rate.
    9. Repeat by going to Step 4 till PM is active.
End
```

ETHERNET PERFORMANCE TUNING

The Ethernet performance management model considered for soft-failure identification is further extended to demonstrate the learning capability *i.e.*, improving the communication network behavior in case of performance degradation.

The parameters considered for improving the network performance are (i) arrival probability; (ii) basic retry slot; (iii) maximum packet length; and (iv) minimum packet length. Besides the parameter's value, the tuning algorithm accepts performance degradation and average number of packets queued up in each of the source buffers. The output is the recommended value of the parameters.

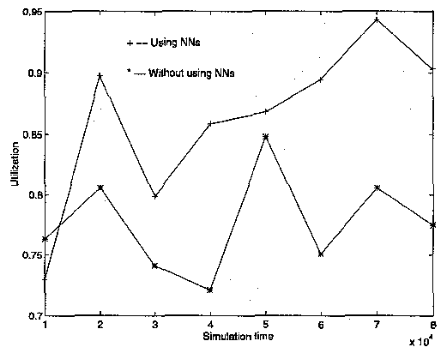

Fig. 4. Performance improvement on using neural network.

The simulation model is developed in C++ programming language on Sun Sparc 20 workstation, running SUN OS 4.2. Each of the "actor" and the "critic" neural networks is trained by using *backpropagation* learning procedure with 5 neurons in the input, 4 neurons in the hidden layer and one neuron in the output layer. To demonstrate the proposed model for network performance tuning, the Ethernet model is simulated both in the absence of neural networks and after integrating with neural networks. The performance of the Ethernet at several instances of simulation are presented. The improvement can be attributed to the adaptability of the neural networks to learn the behavior of the Ethernet and make appropriate changes to result in better performance.

From Figure 4, it can be observed that, there is a substantial improvement in the performance (by over 5%, on an average) when the connectionist model for performance tuning is employed. Initial dip in performance is because of the neural network training to reach the expected level of tuning. This may be overcome by providing the expert knowledge to the neural network before it is placed on the Ethernet for performance tuning.

CONCLUSION

In this paper, network performance problem is presented in two parts. First part identifies the causes of network performance degradataion and the second gives necessary tuning to bring the performance back to normalcy using reinforcement learning model. The model is tested with the hypothetical Ethernet and the results show a significant improvement of over 5%.

References

[1] A. G. Barto, R. S. Sutton and C. W. Anderson, Neuronlike Elements that can Solve Difficult Learning Control Problems, *IEEE Trans. on Systems, Man and Cybernetics*, 13, pp.835-846, 1983.

[2] D. R. Boggs, J. C. Mogul and C. A. Kent, Measured Capacity of an Ethernet : Myths and Reality, *Computer Communication Reveiw*, pp.222-234, 1992.

[3] F. E. Feather, Fault Detection in an Ethernet Network via Anomaly Detectors, *Doctoral Desssertation*, Carnegie Mellon University, July 1992.

[4] F. E. Feather, D. Slewlorek and R. Maxion, Fault Detection in an Ethernet Network Using Anomaly Signature Matching, *Computer Communication Reveiw*, pp.279-288, 1993.

[5] J. P. Hansen, The Use of Multi-Dimensional Parametric Behavior of a CSMA/CD Network for Network Diagnosis, *Ph.D. thesis*, Department Electrical and Computer Engineering, Carniegie Mellon University, 1992.

[6] S. Hayes, Analyzing Network Performance Management, *IEEE Communication Magazine*, 31(5), pp.52-59, 1993.

[7] R. M. Metcafe and D. R. Boggs, Ethernet : Distributed Packet Switching for Local Computer Networks, *Communications of ACM*, 19(7), pp.395-404, 1976.

[8] S. S. Keerthi and B. Ravindran, A Tutorial Survey of Reinforcement Learning, to appear in : *Sadhana*.

[9] G. P. Kumar and P. Venkataram, Network Performance Management Using Realistic Abductive Reasoning Model, in : *IEEE/IFIP International Symposium on Integrated Network Management*, Santa Barbara, California, pp.187-198, May 1995.

[10] G. P. Kumar and P. Venkataram, Network Fault and Performance Management Using Realistic Abductive Reasoning Model, *Journal of Indian Institute of Science*, Special issue on Intelligent Systems, Vol.28, No.6, pp.703-707, 1995.

[11] G. P. Kumar, Integrated Network Management Using Extended Blackboard Architecture, *Doctoral Dissertation*, Indian Institute of Science, Bangalore, July 1996.

[12] G. Vijaykumar, J. A. Fraklin and H. Benbrahim, Acquiring Robot Skills via Reinforcement Learning, *IEEE Control Systems*, pp.13-24, February 1994.