Architecture of Generalized Network Service Anomaly and Fault Thresholds

Zheng Zhang[1], Constantine Manikopoulos[1], and Jay Jorgenson[2]

[1]ECE Department, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA
zxz9622@njit.edu, manikopoulos@adm.njit.edu
[2]Department of Mathematics, CCNY, Convent Ave. at 138 ST., New York, NY 100031, USA
jjorgenson@mindspring.com

Abstract. In this paper we introduce GAFT (Generalized Anomaly and Fault Threshold), featuring a novel system architecture that is capable of setting, monitoring and detecting generalized thresholds and soft faults proactively and adaptively. GAFT monitors many network parameters simultaneously, analyzes statistically their performance, combines intelligently the individual decisions and derives an integrated result of compliance for each service class. We have carried out simulation experiments of network resource and service deterioration, when increasingly congested in the presence of class-alien traffic, where GAFT combines intelligently, using a neural network classifier, 12 monitored network performance parameter decisions into a unified result. To this end, we tested five different types of neural network classifiers: Perceptron, BP, PBH, Fuzzy ARTMAP, and RBF. Our results indicate that BP and PBH provide more effective classification than the other neural networks. We also stress tested the entire system, which showed that GAFT can reliably detect class-alien traffic with intensity as low as five to ten percent of typical service class traffic.

1 Introduction

Network faults can be classified into two types: *hard failures*, in which the network, or some of its elements, are not able to deliver any traffic at all; *soft failures*, network/service anomaly or performance degradation in various performance parameters, i.e., decrease in bandwidth, increase in delay, etc. A hard fault can be easily noticed by all, the system administrators as well as the users. However, defining and otherwise characterizing and detecting soft faults is difficult. Wireless networks are particularly vulnerable to soft faults in that they have much lower bandwidth than their wired counterparts, while they are more prone to overloads, noise, congestion, etc.

In the past few years, some research progress has been made in soft fault detection. A proactive fault management system is presented in [3]. In [2] a study was carried out of path failure detection. In [4], neural networks are applied in billing fraud detection in telecommunication voice networks. Performance anomaly detection in Ethernet is discussed in [5]. In [1]. Cabreta et. al. described their practice of using Kolmogorov-Smirnov (K-S) statistics to detect Denial-of-Service and Probing attacks.

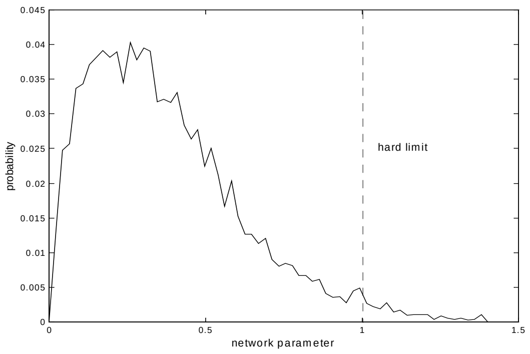

Fig. 1. PDF of a Network Parameter

In this paper, we propose a hierarchical, multi-tier, multi-window, soft fault detection system for both wireless and wired networks, which operates automatically, adaptively and proactively. The system uses statistical models and neural network classifiers to detect anomalous network conditions. The statistical analysis bases its calculations on PDF algebra, in departure from the commonly used isolated sample values or perhaps their averages. The use of PDFs is much more informative, allowing greater flexibility than just using averages. The sample PDF of some network performance parameter, shown in Fig. 1, helps illustrate that point. The figure depicts a hard service limit at some parameter value, drawn as a vertical line. Such a limit might apply for a real-time service, for example, the total packet delay parameter for packetized voice. From the point of view of the hard limit, the network condition described by the PDF in this figure would represent a service failure, in that some packets exceed the limit.

However, the PDF shows that the totality of such failing packets, those in the tail of the PDF to the right of the limit, may be small. Thus, if in fact, this total is smaller than the maximum packet loss rate specification, the service may remain in compliance to the delay limit simply by discarding the laggard packets. Our system generalizes further by combining information of the PDFs of the monitored performance parameters, either all of them or subgroups of them, in one integrated and unified decision result. This combining is powerful in that it achieves much higher discrimination capability that enables the monitoring of individual service classes in the midst of general traffic consisting of all other classes. It is also capable of discriminating against intrusion attacks, known or novel. In fact, network intrusion detection is one of the promising applications of this technology. Others, include transaction oriented e-commerce networks, that typically support large numbers of service classes concurrently, and, as mentioned, various wireless networks, where monitoring the service classes may be challenging due to adverse network conditions.

GAFT gathers data from network traffic, the system log and hardware reports; it statistically processes and analyzes the information, detects network anomaly and failures, for each parameter individually, or in combined groups using neural network classification; it generates the system alarms and event reports; finally, GAFT updates the system profiles based on the newly observed network patterns and the system outputs (see Fig. 2).

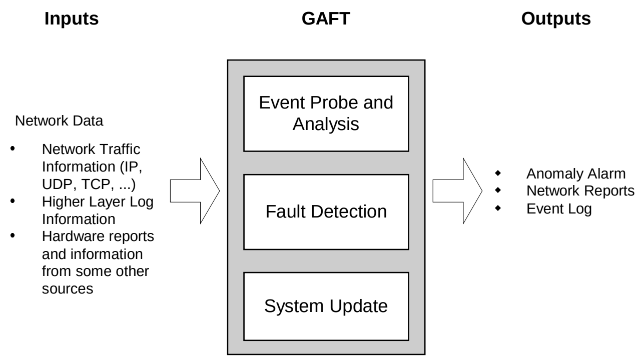
Inputs
GAFT
Outputs
Network Data
Network Traffic Information (IP, UDP, TCP, ...)
Higher Layer Log Information
Hardware reports and information from some other sources
Event Probe and Analysis
Fault Detection
System Update
Anomaly Alarm
Network Reports
Event Log

Fig. 2. The Inputs and Outputs of GAFT

The rest of the paper is organized as follows. The system architecture is outlined in Section 2. Section 3 introduces the statistical algorithms that we are using. Section 4 describes the five neural networks we tested. The simulation environment is given in Section 5. The experimental results using neural network classifiers are presented in section 6. Section 7 reports the results of stress testing on GAFT. Section 8 draws some conclusions and outlines future work.

2 System Architecture

Our system is a distributed application, deployed hierarchically, which consists of several tiers, with each tier containing several Fault Management Agents (FMAs). FMAs are management components that monitor the activities of a host or the network it is attached to. Different tiers correspond to different network scopes that are monitored by the agents affiliated to them.

Generally speaking, each FMA collects network status reports from FMAs one tier below and combines them with data that it collects from its own system logs, as well

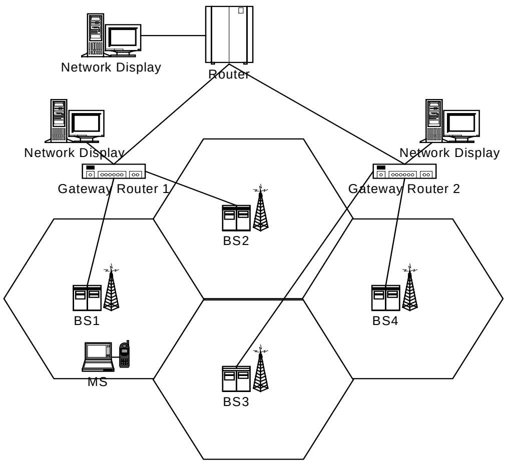

Fig. 3. Sample Wireless Network

as its monitoring of the network; it then generates a network status report, that it sends to the FMA of the next higher tier.

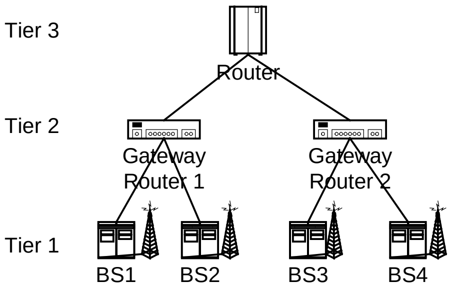

Fig. 4. System Hierarchy

For the sample IP wireless network shown in Fig. 3, the fault detection system can be divided into 3 tiers. Tier 1 agents monitor system activities of the base stations, as well as the nodes in the subnet that the base station supports; they also periodically generate reports for the next higher tier, i.e. Tier 2, agents. Tier 2 agents combine network data that they gather within their cell, based on the network traffic that they observe, as well as the reports they receive from the Tier 1 agents, to formulate their network status decision; they report this to Tier 3 agents. Tier 3 agents collect reports from Tier 2 agents and combine them with their own network observation data, to generate status reports to send to the next higher tier, and so on, tier by tier. The system hierarchy is shown in Fig. 4.

The FMAs of all tiers have the same structure, thus simplifying the system. A diagram of an FMA is illustrated in Fig. 5, which consists of the following components: the probe, the event preprocessor, the statistical processor, the neural network classifier and the post processor. The functionalities of these components are described below:

- *Probe:* Collects data of host activity or associated network behavior, abstracts the data into a set of statistical variables to reflect the network status, and periodically generates reports to the *event preprocessor*.
- *Event Preprocessor:* Receives reports from both the *probe* and *FMAs* of the next lower tiers, and converts the information into the format required by the *statistical model*.
- *Statistical Processor:* Maintains a reference model of the typical network and host activities, compares the reports from the *event preprocessor* to the reference models, and (1) generates a decision for one or each of the monitored parameters individually, or (2) forms a stimulus vector that combines all or groups of parameters to feed into the neural network classifiers.
- *Neural Network Fault Classifier:* Processes the stimulus vector from the *statistical model* to decide whether the network status and traffic adhere to the service requirements for each class. Section 4 introduces the neural network classifiers used in the system in more detail.
- *Post Processor:* Generates reports for the agents at higher tiers. At the same time, it may display the results through a user interface.

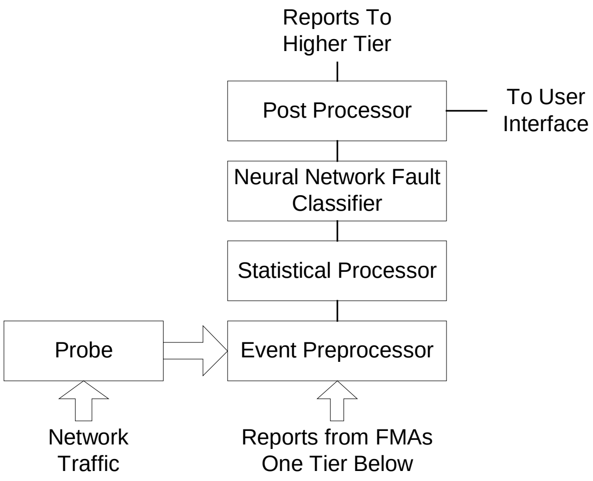

Fig. 5. Fault Management Agent

Because network traffic is not stationary and network soft faults may have different time durations, varying from a few seconds to several hours or longer, we need a statistical model, which is capable of efficiently monitoring network traffic with different simulation observation time windows. Based on the above, we designed and deployed a layer-window statistical model, shown in Fig. 6, with each layer-window corresponding to a monitoring time slice of increasing size.

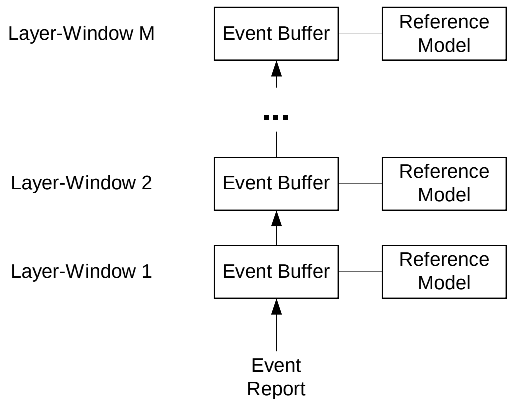

Fig. 6. Statistical Model

The newly arrived events will first be stored in the event buffer of layer 1. The stored events are compared with the reference model of that layer and the results are then fed into the neural network classifier to decide the network status during that time window. The event buffer will be emptied once it becomes full, and the stored events will be averaged and forwarded to the event buffer of layer 2. This process will be repeated recursively until the top level is reached where the events will simply be dropped after processing.

3 Statistical Algorithm

Statistical methods have been used in fault management systems to detect anomaly network activities; however, most of these systems simply measure the means and the variances of some variables and detect whether certain thresholds are exceeded. SRI's NIDES [7][8] developed a more sophisticated statistical algorithm by using a χ^2-like test to measure the similarity between short-term and long-term profiles. Kolmogrov-Smirnov (K-S) statistics is used in [1] to model and detect anomaly traffic patterns.

Our current statistical model uses a similar algorithm as the above two algorithms but with major modifications. Therefore, we will first briefly introduce some basic information about the χ^2-like test and the KS test.

3.1 χ^2-Like Test

In NIDES, user profiles are represented by a number of probability density functions (PDFs). Let S be the sample space of a random variable and events E_1, E_2,…, E_k a mutually exclusive partition of S. Assume P_i is the expected probability of the occurrence of the event E_i, and let $P_i^{'}$ be the frequency of the occurrence of E_i during a given time interval. Let N denote the total number of occurrences. The NIDES statistical algorithm used a χ^2-like test to determine the similarity between the expected and actual distributions through the statistic:

$$Q = N \times \sum_{i=1}^{k} \frac{(P_i' - P_i)^2}{P_i} \tag{1}$$

When N is large and the events E_1, E_2,…,E_k are independent, Q approximately follows a χ^2 distribution with $(k-1)$ degrees of freedom. However in a real-time application the above two assumptions generally cannot be guaranteed, thus, empirically, Q may not follow a χ^2 distribution. NIDES solved this problem by building an empirical probability distribution for Q, which is updated daily in a real-time operation.

3.2 Kolmogrov-Smirnov Statistic

In using the Kolmogrov-Smirnov test, the reference models and the observed system activities are represented by a number of cumulative density functions (CDF). The equation to compare a theoretical, completely specified target cumulative distribution function (CDF), $F_T(x)$, and a sample CDF, $F_s(x)$, is

$$D = \max_{-\infty<x<\infty} \left| F_T(x) - F_S(x) \right| \tag{2}$$

What makes the K-S statistic powerful is the fact that, in the case of the null hypothesis (data sets drawn from the same distribution), the distribution of D is independent of $F(x)$ and can be calculated with useful accuracy. This is very important property for network monitoring, when little may be known about the underlying distribution for either the typical or the anomalous traffic.

There are several variants on the K-S tests. Among them, the Anderson-Darling statistic calculates a weighted K-S measure:

$$D^* = \max_{-\infty<x<\infty} \frac{\left| F_T(x) - F_s(x) \right|}{\sqrt{F_T(x).[1 - F_T(x)]}} \tag{3}$$

As another example, Kuiper's statistic is the sum of the maximum distance of $F_T(x)$ above and below $F_s(x)$.

$$\begin{aligned} V &= D_+ + D_- \\ &= \max_{-\infty<x<\infty x} [F_T(x) - F_s(s)] + \max_{-\infty<x<\infty x} [F_s(x) - F_T(s)] \end{aligned} \tag{4}$$

3.3 The Algorithms Used in GAFT

Similarly to the approach taken by NIDES, in our system, the network activities are sampled and abstracted into PDFs. However, since we are using a neural network fault classifier, that is capable of learning from examples by training, to discern network degradations, we are not so concerned with the actual distribution of *Q*. Nevertheless, because network traffic is not stationary and network faults may have different time durations, the algorithm must be reliable in detecting the differences between the observed and the reference PDFs, so as to adapt effectively to the observed traffic patterns.

The similarity-measuring algorithm that we are using is shown below:

$$Q = f(N).[\sum_{i=1}^{k} |p_i^{'} - p_i| + \max_{i=1}^{k} (|p_i^{'} - p_i|)] \tag{5}$$

where *f*(*N*) is a function that takes into account the total number of occurrences during a time window. This similarity algorithm may be interpreted as incorporating the KS statistic in equation (2) along with the area of the difference between the target and sample PDF curves.

Besides similarity measurements, we also designed an algorithm for the real-time updating of the reference model. Let $\overline{p}_{old}$ be the reference model before updating, $\overline{p}_{new}$ be the reference model after updating, and $\overline{p}_{obs}$ be the observed user activity within a time window. The formula to update the reference model is

$$\overline{p}_{new} = s \times \alpha \times \overline{p}_{obs} + (1 - s \times \alpha) \times \overline{p}_{old} \tag{6}$$

where

α	is the predefined learning rate
s	is the dynamic adaptation step based on the output of the neural network classifier.

Assume that the output of the neural network classifier is a continuous variable *u* between –1 and 1, where –1 means fault with absolute certainty and 1 means normalcy again with complete confidence. In between, the values of *u* indicate proportionate levels of certainty. The function for calculating *s* is

$$s = \begin{cases} u, & \text{if } u \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Through the above equations, we ensured that the reference model would be updated actively for typical traffic while kept unchanged when failures occurred. The fault events will be diverted and stored for future neural network learning.

4 Neural Network Fault Classifiers

Neural networks are widely considered as an effective approach to classify patterns. In [10], BP neural networks were used to detect anomalous user activities. Jiang et al [6]

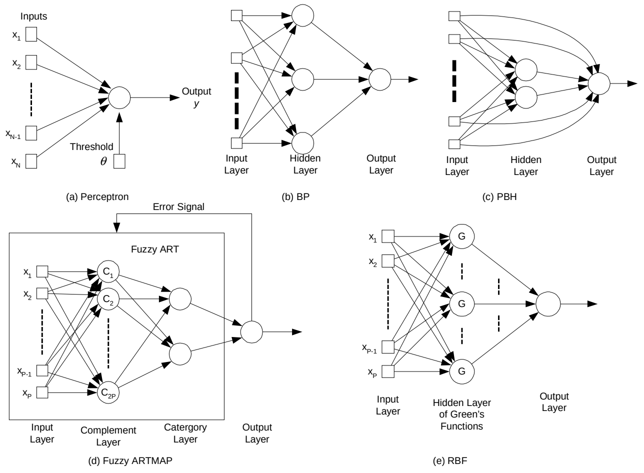

Fig. 7. Neural Network Architectures

built a network fault management system using artificial intelligence technologies. In order to comprehensively investigate the performances of neural network classifiers, we examined five different types of neural networks: Perceptron, BP, PBH, Fuzzy ART MAP and RBF.

The perceptron [11], Fig. 7(a), is the simplest form of a neural network used for the classification of *linearly separable* patterns. Although our data sets will not, in general, be linearly separable, we are using the perceptron as a baseline to measure the performances of other neural networks.

The Backpropagation network [11], or BP, Fig. 7(b), is a multi-layer feedforward network. BPs have strong generalization capabilities and have been applied successfully to solve a variety of difficult and diverse problems. Here we tested BP networks with the number of hidden neurons ranging from 2 to 8.

The Perceptron-backpropagation hybrid network [9], or PBH, Fig. 7(c), is a superposition of a perceptron and a small backpropagation network. PBH networks are capable of exploring both linear and nonlinear correlations between the input stimulus vectors and the output values. We tested PBH networks where the number of hidden neurons ranged from 1 to 8.

The Fuzzy ARTMAP [12] in its most general form is a system of two Fuzzy ART networks ART_a and ART_b whose F2 layers are connected by a subsystem referred to as a “match tracking system”. We are using a simplified version of Fuzzy ARTMAP [13], Fig. 7(d), which is implemented for classification problems. We tested ARTMAP networks with the number of category neurons ranging from 2 to 8.

The Radial-basis function network [11], or RBF, Fig. 7(e), involves three entirely different layers. We tested RBF networks with hidden neurons ranging from 2 to 8.

5 The Simulation Environment

To validate our statistical models and to test the system architecture, we built a virtual network using simulation tools. The experimental testbed that we built using OPNET, a powerful network simulation facility, is shown in Fig. 8. The testbed is a 10-BaseX LAN that consists of 11 workstations and 1 server. This might correspond to the lower tier (Tier 1) only of the system hierarchy diagram shown in Fig. 4. It adequately illustrates the performance of a typical GAFT agent at Tier 1, as well as provides insight for the GAFT agent to be basically the same at all tiers.

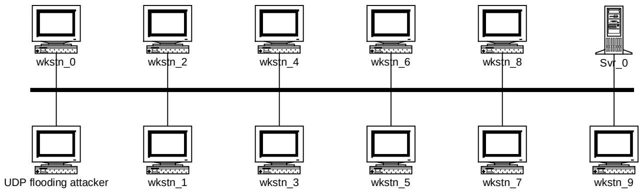

Fig. 8. Simulation Testbed

In the real network environment, a network/service degradation or failure may be caused by equipment malfunctions, transmission media breakdowns, malicious external or internal attacks, etc. In this work, we simulated network condition degradation by injecting high volumes of class-alien UDP traffic into the simulation testbed. This causes increased delays and may lead to decreased throughput and increased packet loss rates. The simulation generates class-normal traffic using standard driving distributions, such as Poisson, uniform, constant, etc. The class-alien traffic is also simulated using the standard distributions but different parameters. Other methods of simulating network degradation can be employed as well. Because in Ethernet all classes of traffic are transmitted over the same media, an anomaly behavior of one class will interfere with other network services. In other words, the behavior of different classes of traffic becomes correlated, as is often the case in many real-life networks.

To extensively test the performances of the system, we ran several independent scenarios with different typical and anomaly traffic loads. For each simulation scenario, we collected 10,000 records of network traffic. We divided these data into two separate sets, one set of 6000 records for training and the other of 4000 records for testing. The training records included typical as well as traffic tainted by the injected alien packets and thus labeled. In each scenario, the system was trained for 100 epochs.

6 Results on Neural Network Classifiers

We evaluated the performance of each of the neural networks based on the *misclassification rates* of the outputs. The misclassification rate is defined as the percentage of the network traffic that is misclassified by neural network fault classifiers during one epoch, which includes both *false positive* and *false negative* misclassifications.

TABLE 1 lists the traffic loads of the four simulation scenarios that we ran for neural network testing.

Table 1. Traffic Loads of the Four Simulation Scenarios

	Typical Class Traffic	Class-Alien Traffic
Scenario 1	600kbps	50kbps
Scenario 2	600kbps	100kbps
Scenario 3	2Mbps	50kbps
Scenario 4	2Mbps	100kbps

In the rest of this section, we will present and analyze the simulation results of the neural network classifiers one by one.

6.1 Perceptron

The mean squared root errors and the misclassification rates of the perceptrons within the four simulation scenarios are tabulated in TABLE 2.

Table 2. The simulation results for perceptrons

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Misclassification rate	0.167	0.202	0.234	0.119

We can see that the perceptron performed poorly in all the four scenarios: Mean squared root errors are between 0.6 and 0.7; and misclassification rates are between 0.1 and 0.2. Both the MSR errors and the misclassification rates are unacceptably high for a fault detection system.

6.2 Fuzzy ARTMAP and RBF

The results of Fuzzy ARTMAP and RBF nets are shown in Fig. 9. The x-axis values of the figures represent the number of category neurons in Fuzzy ARTMAP and the hidden neurons in RBF. The y-axis values represent the lowest Misclassification Rates that these neural classifiers achieved within the 100 epochs.

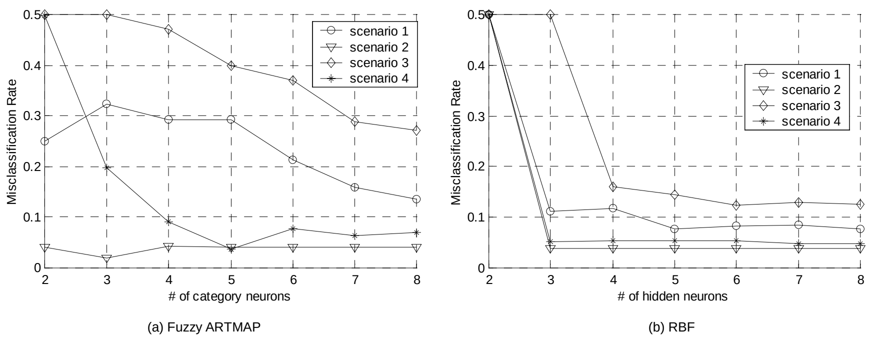

Fig. 9. Results of Fuzzy ARTMAP and RBF

From the above figures, we can see that, as the number of hidden neurons increases, the performance of both, ARTMAP and RBF networks, improves. In most of the cases, both of them outperformed the perceptron.

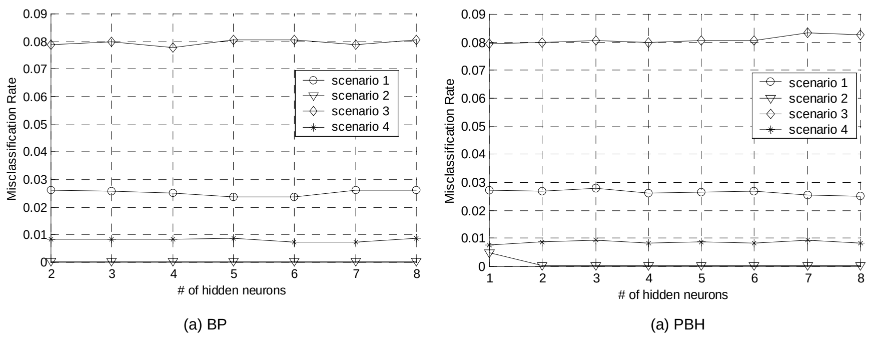

Fig. 10. Results of BP and PBH

6.3 BP and PBH

The results of BP nets are illustrated in Fig. 10. The figures indicate that BP and PBH networks have similar performance, and that both neural networks consistently perform better than the other three types of neural networks. The curves in these figures are flat: the MSR errors and misclassification rates do not decrease as the number of hidden neurons increases. We believe the reason is that, the stimulus vectors provided to the neural network classifier are very powerful and telling, especially in the sense that multiple performance parameter K-S similarity measures are combined together in one integrated pattern. Thus, there is ample room for the classifier to undertake more challenging network conditions.

7 Stress Testing the GAFT System

In this section, we will stress test the sensitivity, and thus effectiveness, of our system. We simulated various network fault scenarios using the test bed we introduced in section 5. The traffic loads of the simulations we ran for stress testing are specified in TABLE 3. From section 6, we can see that BP and PBH performed the best among the five neural networks tested, and that, for these two neural networks, increment in the number of hidden neurons did little to help with their performances. Therefore, the neural network that we chose for stress testing was a BP network with 2 hidden neurons.

Table 3. The traffic loads for stress testing

Typical-Class Traffic	Class-Alien Traffic
600 Kbps	10Kbps, 20Kbps, 30Kbps, 40Kbps, 50Kbps, 70Kbps, 100Kbps, 200Kbps
2Mbps	10Kbps, 50Kbps, 100Kbps, 150Kbps, 200Kbps

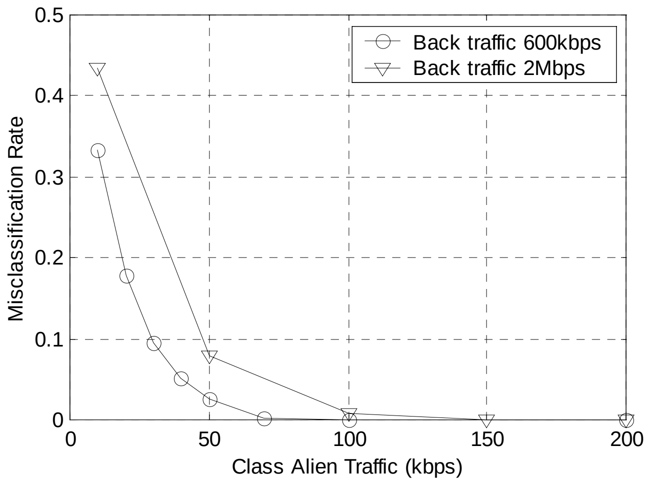

Fig. 11. The results of Stress Testing

Through the simulations, we expect to see the changes in *the Misclassification Rates* of the system as functions of the typical and anomaly traffic volumes.

The results of stress testing are shown in Fig. 11. From the figure, we can see that both the MSR errors and the misclassification rates decrease as the class-alien traffic level increases. This is because traffic patterns of higher-volume foreign traffic yield greater differences from the reference model than those created by lower-volume. We can also notice that, for a certain class-alien traffic level, the performance for the 600Kbps typical traffic background is consistently better than that of 2Mbps, as one may reasonably expect.

Fig. 12 shows the ROC (Receiver Operating Characteristic) curves of some selected simulation scenarios. The x-axis of the figure is the false alarm rate, which is the rate

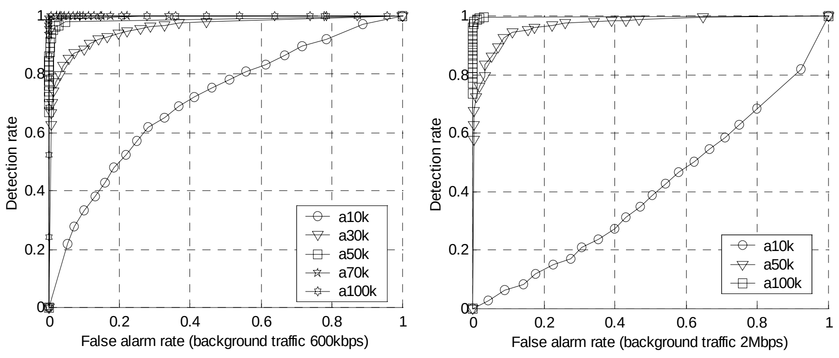

Fig. 12. ROC Curves

of the typical traffic events being classified as faults or anomalies; the y-axis of the figure is the detection rate, which is calculated as the ratio between the number of correctly detected faults/anomalies to their total number. For each curve, the point at the upper left corner represents the optimal detection with high detection rate and low false alarm rate. From the figure, we can observe the same tendency as in Fig. 11: The detection performance improves as the alien traffic intensity increases. In fact, we can see that, when the alien traffic level is 70Kbps for 600Kbps typical class traffic and 100Kbps for 2Mbps typical traffic, the system performance approaches the optimum.

8 Conclusions

In this paper, we introduced the framework of a hierarchical, multi-tier, multi-window fault detection system, using statistical preprocessing and neural network classification. We described our experiments using five different neural network classifiers. The results showed that BP and PBH nets outperform Perceptron, Fuzzy ARTMAP and RBF networks. Thus, the classification capabilities of BP and PBH classifiers are more desirable for statistical anomaly fault detection systems. We also presented some of the results of stress testing. The results indicate that the system is very efficient. It can reliably detect soft faults with traffic anomaly intensity as low as five to ten percent of the typical service class traffic intensity.

Acknowledgements. Our research was partially supported by a Phase I SBIR contract with US Army.

We would also like to thank OPNET Technologies, Inc^{TM}, for providing support for the OPNET simulation software.

References

[1] Joao B.D. Cabrera, B. Bavichandran, R.K. Mehra, "Statistical Traffic Modeling for Network Intrusion Detection", *Proceedings of 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication systems,* Aug. 2000, pp. 466-473.

[2] N. Dawes, J. Altoft, and B. Pagurek, "Network Diagnosis by Reasoning in Uncertain Nested Evidence Spaces", *IEEE Trans. Commun.*, vol. 43, 1995, pp. 466.

[3] L.L. Ho, D.J. Cavuto, S. Papavassiliou, A.G. Zawadzki, "Adaptive and Automated Detection of Service Anomalies in Transaction-oriented WANs: Network Analysis, Algorithms, Implementation, and Deployment", *IEEE Journal on Selected Areas in Commun.*, vol. 18, May 2000, pp. 744-757.

[4] J.R. Dorronsoro, F. Ginel, and C. Sanchez, "Neural fraud Detection in Credit Card Operations", *IEEE Trans. Neural Networks*, vol. 8, 1997, pp. 827.

[5] R. Maxion and F.E. Feather, "A Case Study of Ethernet in a Distributed Computing Environment", *IEEE Trans. Reliability*, vol. 39, Oct. 1990, pp. 433-443.

[6] S. Jiang, D. Siboni, A.A. Rhissa, G. Beuchot, "An Intelligent and Integrated System of Network Fault Management: Artificial Intelligence Technologies and Hybrid Architectures", *Proceedings of IEEE Singapore International Conference on Networks,* July 1995, pp. 265-268.

[7] A. Valdes, D. Anderson, "Statistical Methods for Computer Usage Anomaly Detection Using NIDES", *Technical report*, SRI International, January 1995.

[8] H. S. Javitz, A. Valdes, "The NIDES Statistical Component: Description and Justification", *Technical report,* SRI International, March 1993.

[9] R. M. Dillon, C. N. Manikopoulos, "Neural Net Nonlinear Prediction for Speech Data", *IEEE Electronics Letters*, Vol. 27, Issue 10, May 1991, pp. 824-826.

[10] A.K. Ghosh, J. Wanken, F. Charron, "Detecting Anomalous and Unknown Intrusions Against Programs", *Proceedings of IEEE 14th Annual Computer Security Applications Conference,* 1998, pp. 259 –267.

[11] Simon Haykin, *Neural Network A Comprehensive Foundation*, Macmillan College Publishing Company, 1994.

[12] G.A. Carpenter, et al, "Fuzzy ARTMAP: An adaptive resonance architecture for incremental learning of analog maps", *International Joint Conference on Neural Networks,* June 1992.

[13] NeuraWare Inc., Neural Computing A Technology Handbook for NeuralWorks Professional II/PLUS and Neural Works Explorer, NeuralWare Inc., 1998.