AUTOMATED NETWORK FAULT MANAGEMENT *

J.S. Baras, M. Ball, S. Gupta, P. Viswanathan, and P. Shah

Center for Satellite and Hybrid Communication Networks
Institute for Systems Research
University of Maryland
College Park, Maryland 20742

ABSTRACT

Future military communication networks will have a mixture of backbone terrestrial, satellite and wireless terrestrial networks. The speeds of these networks vary and they are very heterogeneous. As networks become faster, it is not enough to do reactive fault management. Our approach combines proactive and reactive fault management . Proactive fault management is implemented by dynamic and adaptive routing. Reactive fault management is implemented by a combination of a neural network and an expert system. The system has been developed for the X.25 protocol. Several fault scenarios were modeled and included in the study: reduced switch capacity, increased packet generation rate of a certain application, disabled switch in the X.25 cloud, disabled links. We also modeled occurrence of alarms including severity of the problem, location of the event and a threshold. To detect and identify faults we use both numerical data associated with the performance objects (attributes) in the MIB as well as SNMP traps (alarms). Simulation experiments have been performed in order to understand the convergence of the algorithms, the training of the neural networks involved and the G2/NeurOn-Line software environment and MIB design.

INTRODUCTION

Fault management [14, 15, 16, 17] includes detecting, isolating, and repairing problems in the network, tracing faults, given many alarms in the system, using error logs and tracing errors through the log reports [1]. One of the problems faced by network control centers is that of handling extremely large volumes of data dealing with the performance of the networks. The data volume makes the task of finding the problem a very time-consuming process. However, unlike some of the other network management functions listed in the ISO model, in fault management, speed is very crucial and recovery from a problem has to occur quickly.

Several efforts have taken place to tackle the fault management problem, some of which are described in [4, 5, 6, 7, 8, 9, 10]. Although several interesting issues have been addressed in these papers, such as trouble ticketing and alarm correlation, most of the work has been done through the use of expert systems alone [12, 13] without the use of neural networks. Furthermore, in these sources, we have not seen the use of SNMP statistics for the fault management problem.

The expert system approach to diagnosis is intuitively attractive, as symptoms can be linked to causes explicitly, in a rule-based knowledge representation scheme. The limitations of rule-based expert systems are revealed when they are confronted with novel fault situations for which no specific rules exist. Novel faults, for which the neural network has not been trained, or for which no output neuron has been assigned, are generalized and matched to the closest fault scenario for which the network has been trained. Each approach contains its own strengths and weaknesses. In order to take advantage of the strengths of each technique, as well as avoiding the weaknesses of either we used an integrated neural network/expert system diagnostic strategy [2, 3].

Dynamic fault management is a critical element of network management. It is even more difficult in military networks because in addition to hard faults and soft faults (caused by performance degradation) we also have faults caused by the varying situation and scenario of the battle. Future military communication networks will have a mixture of backbone terrestrial, satellite and wireless terrestrial networks. The speeds of these networks varies and they are very heterogeneous. As networks become faster, it is not enough to do reactive fault management. Our approach combines proactive and re-

*This work was supported in part by the U.S. Department of the Army, Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002, Federated Laboratory ATIRP Consortium, in part by the Center for Satellite and Hybrid Communication Networks under NASA cooperative agreement NCC3-528, and in part by a grant from Space Systems Loral.

active fault management . Proactive fault management is implemented by dynamic and adaptive routing. Reactive fault management is implemented by a combination of a neural network and an expert system.

In the work reported here we concentrate on fault management at the application level. Each application generates packets using a Markov Modulated Poison Process (MMPP). We assume two packet priorities, and non-preemptive queue management. The system has been developed for the X.25 protocol. The dynamic routing is based on dynamically adjustable link costs on the basis of utilization to induce correction via rerouting based on minimum cost. In our model the following performance data are collected by the network: blocking of packets, queue sizes, packets throughput from all applications, utilization on links connecting subnetworks, end-to-end delays experienced by packets. Several fault scenarios were modeled and included in the study: reduced switch capacity, increased packet generation rate of a certain application, disabled switch in the X.25 cloud, disabled links. These scenario are used to train the neural network, so as to predictively recognize the genesis of faults.

We also modeled occurrence of alarms including severity of the problem, location of the event and a threshold. Decisions of whether or not to send an alarm are determined by examining data over a user-specified time window. We implemented components of SNMP monitoring based on RFC 1382, including agents and traps [18, 19, 20]. We have completed a small prototype demonstration system which consists of: an OPNET simulation of a network and its faults, a MIB, and a tightly coupled Neural Network and Expert System. We used neural networks based on radial basis functions. To detect and identify faults we use both numerical data associated with the performance objects (attributes) in the MIB as well as SNMP traps (alarms). Performance data from the X.25 network is supplied as input to the neural network and data concerning SNMP statistics, SNMP traps, and alarms are supplied as input to the expert system. We are using both neural networks and expert systems since all faults cannot be explained through the use of just alarms or SNMP traps. An overview of the system is shown in Figure 1.

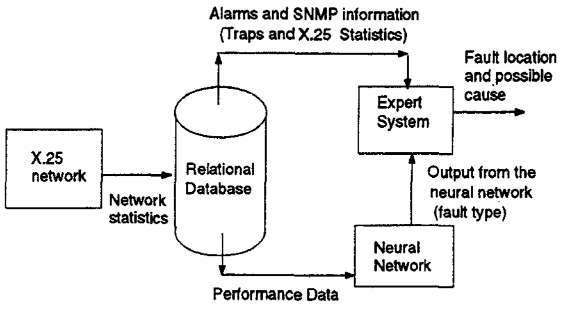
Alarms and SNMP information (Traps and X.25 Statistics)
Fault location and possible cause
Expert System
X.25 network
Relational Database
Network statistics
Output from the neural network (fault type)
Neural Network
Performance Data

Figure 1: The overall system

NETWORK TOPOLOGY AND SIMULATION

The network that we have simulated in OPNET is based on the X.25 protocol. Each user corresponds to a Data Terminal Equipment (DTE), connected to a Data Communications Equipment (DCE). Thus, having 10 users implies having 10 DTE/DCE pairs, where each DTE can have several logical channels. Each DTE can handle 2 applications, thus making it possible to run up to 20 applications at a time. There are both permanent virtual circuits (PVCs) and virtual calls. Four PVCs have been predefined. The PVCs are DTE to DTE connection. In addition to the DTEs and DCEs pertaining to the X.25 model, there is also a SNMP manager.

In our simulation, the X.25 cloud consists of 15 nodes used to transmit the packets in a store-and-forward manner. These 15 nodes are grouped into 3 subnetworks, where each subnetwork consists of 5 nodes. The division of the X.25 cloud into subnetworks should be done so that a single neural network can be appropriately assigned to monitor each subnetwork. A typical subnetwork is shown in Figure 2.

We have incorporated the following assumptions in the simulation model. Each application generates packets using a Markov Modulated Poisson Process (MMPP). The source sends packets whose sizes are fixed. The MMPP source is used in order to simulate a bursty traffic model for data. The amount of data transfer is established by a random number generator. Each packet has a priority of 0 (low) or 1 (high), depending on the user generating the packet. The input and output queues have finite capacity and fixed service rate that are user-specified. The rate of the input queue corresponds to the switch rate while the rate of the output queue corresponds to the link rate. There is a collection of source/destination DTE pairs. The association between DTEs is many-to-many, as in electronic mail,

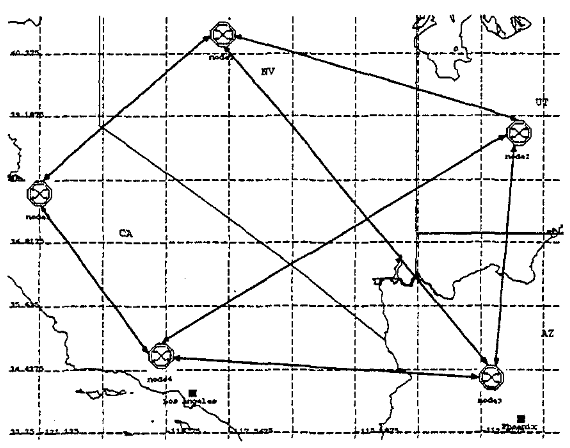

Figure 2: The overall system.

for example. We used queuing with priorities but non-preemptive. All the packets (arriving from the various nodes via the input links) are inserted into one queue.

We performed simulations with widely different traffic patterns. When running a simulation for T seconds, we are varying the traffic in such a way that there are periods when traffic is high and other periods when traffic is light. The performance data being collected consist of statistics about the following parameters: Packet drop rates, queue sizes, packet throughput from all the applications, link utilizations, end-to-end delays experienced by packets. In addition, in the simulation we also have SNMP variables monitored and have implemented traps.

A MINIMUM COST ROUTING ALGORITHM

When a packet is created by an application running on a DTE, it is divided into a number of packets of fixed length (the length can be chosen by the network designer). In our simulation, the length of each packet was 128 bytes, the default value as specified in the X.25 Recommendations. A source to destination pair is then assigned to the message. Based on this pair, a route is selected from the routing table and is assigned to the message based on a minimum cost routing algorithm.

We used minimum cost routing based on dynamically changing link costs, in order to implement some proactive fault management. At the start of the simulation, all the links have zero cost assigned to them. As the simulation progresses, this cost is updated periodically by relating the cost of the link to the utilization on the link using the following cost function:

$$c_i = (1-\alpha)c_{i-1} + \alpha\frac{1}{1-\rho_i}$$

where c_i is the cost of the link at the i^{th} time instant (when the data is sampled), c_{i-1} is the cost of the link at the previous time instant, α is a weighting factor between 0 and 1 (inclusive), and ρ_i is the utilization on the link at the i^{th} time instant. The weighting factor, α, is used in order to take into account the dynamics of the network. The choice of α is left to the network designer. In our simulations, we chose α to be a number greater than 0.5, thus assigning more weight to the current value of the utilization. When the utilization of a link increases, the cost of the link increases also (though not in a linear fashion). As a result, the traffic will be re-routed through links that are relatively underutilized.

In addition to depending on the source and destination addresses, this routing algorithm also depends on the maximum number of hops allowed. This parameter is specified by the user.

FAULT SCENARIOS AND DATA

The modeling of faults is done as follows. We define a **normal** state in the network, where **normal** refers to levels of traffic flow that are not unusually low or high, e.g link utilization between 0.20 and 0.70. Then, a set of fault scenarios are modeled and used to train the neural network system. By training the neural network to understand a normal state of operation, it would then be able to recognize abnormal states also. The fault scenarios that we have simulated are the following:

1. Reducing switch capacity, i.e. dropping the service rate. This would affect dropping of packets and response times for applications.
2. Increase the (normal) packet generation rate of a certain application (e.g. 3 times the original amount of traffic).
3. Disabling of certain switches in the X.25 cloud. This means that the switch is not functional and cannot be used as a hop for a call. Such a fault would cause re-routing of calls via other (working) switches.
4. Disabling certain links.

Alarms

A method for simulating the occurrence of alarms is also incorporated in the simulation. The alarm contains information regarding the severity of the problem, the location of the event (i.e. which node in which subnetwork), and a threshold. The severity levels and alarm codes are: critical (5), major (4), minor (3), warning (2), informational (1), cleared (0). The decision of whether or not to send an alarm is determined by examining the sampled data over a user-specified time window.

SNMP Monitoring

In much of the literature that was reviewed [4, 5, 6, 7, 8], there had been little mention regarding the use of SNMP variables to perform fault management. In our approach, we are logging statistics pertaining to SNMP variables based on the RFC 1382, (SNMP MIB Extension for the X.25 Packet Layer). A list of variables was extracted from RFC 1382 and were logged during the simulation. The subset of variables were chosen from the RFC because they are helpful in identifying faults that could occur in the X.25 simulation. The variables are logged on a *per DTE* basis and not on a *per logical channel* basis. This is implemented by assigning IDs to each DTE.

SNMP Traps

In addition, we also have the facility for agents to send traps to a manager when something goes wrong. Here, an agent refers to a node in the X.25 cloud. This manager is designed to manage the switches in the X.25 cloud. It does not receive traps from the DTEs or DCEs in the network. According to RFC 1215 ("A Convention for Defining Traps for use with the SNMP), there are six basic types of traps, together with a seventh (enterprise-specific) trap. These are : *coldStart(0)*, *warmStart(1)*, *linkDown(2)*, *linkUp(3)*, *authenticationFailure(4)*, *egpNeighborLoss(5)*, *enterpriseSpecific(6)*. In our simulation, we have implemented traps 2, 3, and 6 above.

EXPERT SYSTEMS AND NEURAL NETWORKS

OPNET/NEURONLINE Interface

The data from the X.25 simulation in OPNET is gathered in a flat file and stored in an ORACLE database. The data is then read by G2 and Neuronline, where the former is the expert system and the latter is the neural network component. After careful review of the alternatives we chose radial basis function networks (RBFN) as the neural network architecture for conducting classification. In implementing our system, we used a combination of both neural networks and expert systems.

Radial Basis Function Networks

Recently, researchers have been using radial basis function networks for handling classification problems [3, 11]. RBFNs are three-layered networks, with an input layer, a hidden layer, and an output layer. Unlike backpropagation networks, RBFNs use Gaussian transfer functions, one per hidden node. The hidden nodes have spherical (or elliptical) regions of significant activation. The finite bounding of the activation regions enables RBFNs to detect novel cases. Another advantage of RBFNs is that they require less (typically an order of magnitude) time for training compared to backpropagation networks. However, they have a slower run-time execution [11].

The training of RBFNs is done in three stages. In the first stage, the center of each of the radial basis function units is determined using the *k-means* clustering algorithm. This is an unsupervised technique that places unit centers centrally among clusters of points. In the second stage, the unit widths are determined using the nearest neighbor technique, which ensures the smoothness and continuity of the fitted function. In the final stage, the weights of the second layer of connections are found using linear regression.

Network Monitoring

One of the most crucial elements in performing fault management of networks is *speed* for fault detection, fault location, and identification of the type of fault.

For managing the X.25 network, we used a *hybrid* architecture of neural networks and expert systems to perform the fault management functions. Specifically, we used RBFNs to analyze the *performance* data being generated by OPNET. There is one RBFN for each subnetwork. The size and structure of each subnetwork need not be the same and it is an arbitrary design issue that is left to the network designer. The possible outputs of the neural networks are the different classes of faults that could occur in the X.25 subnetworks. When a fault occurs within a certain subnetwork, the RBFN assigned to monitor that subnetwork will alert the network operator that a fault of a specific class (e.g. disabled node) has occured. However, this will not inform the operator of the *location* of the fault. Thus, in the example above, the operator would know that a node in a specific subnetwork was disabled, but he/she would not know *which* node was disabled. Then, based on the outcome of the neural network, appropriate action is taken by the expert system. The expert system uses information about alarms and SNMP traps, together with the SNMP variables which we chose from RFC 1382, to make its conclusions regarding the possible location and cause of the fault. We implemented special rules to handle disabled nodes, others to handle failed links, and so on.

FIRST LEVEL OF FAULT DETECTION AND DIAGNOSIS: NEURAL NETWORKS

We used one radial basis function network for each subnetwork in the X.25 cloud. In the *training* phase for a specific neural network we used the performance data obtained directly from the network. This data is then scaled using a data rescaler, which was configured to use zero mean, unit variance scaling on the input and no scaling on the output. The scaled data is then used by the trainer to train the RBFN. A fit tester is also available. The criterion chosen for the fit tester is fraction misclassified. Thus, the output of the fit tester is a number between 0 and 1, reflecting how accurately data samples are classified.

The neural network has spherical nodes for its hidden layer. The number of hidden nodes per class was chosen through trial and error, after several training sessions, until the desired performance is achieved. During our experiments, it was found that as the number of hidden nodes increased, the fit tester error decreased (though not linearly), thus implying that there was a better fit of the data by the neural network. However, a higher number of hidden nodes also meant a longer training period. The training of the neural networks is affected by the following factors: The quality of the input data and how well it reflects the conditions of the X.25 network; The number of hidden nodes in the hidden layer of the RBFN; The number of input variables that are supplied to the neural network (we supplied the utilization levels on all the links, the queue sizes, and the measured packet throughput at each node); The discriminating characteristics of data for faults occuring simultaneously.

Since a neural network observes patterns and makes inferences based on those patterns, similar patterns for different fault classes would lead to misclassifications. In several experiments, when one node in a subnetwork was blocked, the average queue size of other nodes in the subnetwork increased drastically, beginning at the time of the node blockage. This occurs as a result of re-routing of the X.25 calls. In such cases, there are certain distinct patterns that help the neural network to identify the different cases. However, there are other instances when it is more difficult. For example, a link failure and a node failure both lead to re-routing of traffic. If the samples of training data are small, it is very difficult for the neural network to distinguish between a node failure and a link failure, simply by analyzing the re-routing that occurs. Thus, more data is needed to distinguish between the re-routing that occurs in these two cases in order to have a small percentage of misclassification; and this was verified by our experiments.

Neural Network Training: 5 Classes					
Total Hidden Nodes	% Error Normal State	% Error Disabled Node	% Error Excess Thrput	% Error Degraded Buffer	% Error Disabled Link
175	0.13	0.16	0.14	0.19	0.21
200	0.10	0.13	0.13	0.17	0.18
210	0.08	0.11	0.12	0.16	0.15
230	0.07	0.10	0.09	0.14	0.11

Table 1: Neural network training chart for the third test.

Since there is no fixed method to train neural networks, we arbitrarily selected a few different test cases to develop a better understanding of how the neural networks were trained. The data obtained from the simulations in OPNET was divided so that two-thirds was used for training and one-third for testing. In the first test, we considered 3 classes (normal, disabled node, and excess user traffic). We used 180 samples of data for the normal class and 90 samples for each of the other two classes. In the second test, we repeated the first test but, changed the number of samples of training data to 180 samples per class and retrained the RBFN networks. Comparison of the results indicates that with more data points per class, the total number of hidden nodes decreases for a certain range of error values for the fit tester.

In the third test, we considered all five classes of faults and trained the RBFNs with different sample sizes. We first trained the RBFN with 150 samples for the normal class and 80 samples for each of the other fault classes, giving a total of 470 points in the training set. The results for this case are shown in Table 1. By looking at the last two columns in each row, it is observed that the percentage error is higher for those two fault classes. This provided the motivation for the next test.

In the fourth and final test, we again considered all five classes of faults and trained the RBFNs with different sample sizes. We trained the RBFN with 180 samples each for the normal, disabled node, and excess user traffic classes. For the remaining two faults classes, we used 320 samples for each class giving a total of 1180 points in the training set. The reason for this is because these two cases do not manifest themselves in an obvious manner through the performance data from

Neural Network Training: 5 Classes					
Total Hidden Nodes	% Error Normal State	% Error Disabled Node	% Error Excess Thrput	% Error Degraded Buffer	% Error Disabled Link
60	0.08	0.07	0.06	0.07	0.08
80	0.06	0.05	0.04	0.04	0.07
100	0.05	0.03	0.03	0.04	0.05
125	0.03	0.04	0.03	0.05	0.04

Table 2: Neural network training chart for the fourth test.

the network. When the training for these two cases was performed with 180 samples per class, the percentage of misclassification was very high (approximately 0.40 for each fault class). On the other hand, when we tried using 500 samples for each of these two classes, the RBFN was *overtrained* and all data points (from the testing data) were classified either as *degraded buffer* or *link failed*. Thus, we had to use an intermediate number of points between these two extreme cases of training and the results for this case are shown in Table 2.

After analyzing the behavior of the network under fault conditions in several experiments, it appears that the network topology influences the neural networks ability to discriminate faults. Since all occurrences of a particular fault class are not identical, several different cases need to be presented to the RBFN for the same fault class. Obviously, this corresponds to longer training sessions for the neural networks. Similar observations were also recorded for the other fault classes.

The output of the neural network is used by a classifier to inform the network operator of the current status of the network; the neural network outputs a fault code. If a certain fault code is observed several times (e.g. K times out of M samples), then the expert system is activated to determine further information about the location and cause of the fault, as described in the next section.

SECOND LEVEL OF FAULT DETECTION AND DIAGNOSIS: EXPERT SYSTEMS

The neural network for each subnetwork analyzes the incoming data and if a state other than a normal one appears to be present, then the expert system makes queries to an ORACLE database to determine further information about the observed fault in the network. Different fault conditions induce different queries, as described below.

Single Faults

To detect a node failure at node i, the algorithm first searches for a SNMP trap. Reception of a trap would solve the problem. If, due to some problem in the network, the trap was not received by the SNMP manager (a feature that exists in our simulation), then we execute a query from the expert system looking for the following condition:

$$\sum_{\forall j s.t. \exists link(i,j)} \rho_{ij} < \epsilon$$

In our implementation, we set $\epsilon = 0.01$.
To confirm the hypothesis, we examine:

1. x25StatCallTimeouts counter at the DTEs that are the "source" part of the source/destination pairs for the DTEs connected to node i.
2. x25StatOutCallFailures, x25StatOutCallAttempts counters at the source DTE.

To detect a user connected to node i that is submitting excess traffic to the network, we look for the following condition:

$$\sum_{\forall j s.t. \exists link(i,j)} \rho_{ij} > \tau$$

To confirm the hypothesis, we check:

1. x25StatOutDataPackets at the DTEs connected to node i.
2. Measured packet throughput at node i.
3. x25StatInDataPackets at the destination DTE, i.e. node i, obtained by checking the source/destination pairs in the case of PVCs.
4. x25StatInCalls at the destination DTE, obtained by checking the source/destination pairs in the case of PVCs.

To detect a degraded switch, the algorithm first searches for a SNMP trap. Reception of a trap would solve the problem. To confirm the hypothesis, we check the following:

1. Alarms corresponding to high queue sizes and/or blocking of packets.
2. High end–to–end delays experienced by packets.

To detect a link failure on link (i, j), the algorithm first searches for a SNMP trap. Reception of a trap would solve the problem. If the SNMP manager does not receive a trap, then we execute a query from the expert system looking for the following condition: find i and j such that

$$\rho_{ij} = 0 \quad \text{and} \quad \rho_{ji} = 0.$$

In addition, we check the x25StatRestartTimeouts and x25StatResetTimeouts counters.

Multiple Faults

In the case of multiple faults, we simply need to examine the outputs of the RBFN neural networks and determine which ones do not correspond to normal traffic. By doing so, we eliminate a large number of nodes in the X.25 network and we can focus on those subnetworks that are experiencing problems. In the work to date, we did not consider multiple faults occurring simultaneously within the same subnetwork since the probability of occurrence of such an event is much smaller than the probability of occurrence of multiple faults within different subnetworks. In the event of faults occurring in one subnetwork, with the resulting effects propagating to another subnetwork, the RBFNs of both subnetworks would indicate problem situations and the results of the queries from both subnetworks would have to be examined.

These strategies were validated with the simulation results from OPNET. One should note that the rules which were constructed for the expert system are driven by the X.25 network architecture. It is possible that different architectures would probably use the above rules with some modifications.

REFERENCES

[1] Kornel Terplan. *Communication Networks Management.* Prentice Hall, 2nd. edition, 1992.

[2] W.R. Becraft and P.L. Lee. An integrated neural network/expert system approach for fault diagnosis. *Computers Chem. Engng*, 17(10):1001-1014, 1993.

[3] James Hendler and Leonard Dickens. Radial basis function networks for classying process faults. *AISB Conference*, April 1991.

[4] Joseph Pasquale. Using expert systems to manage distributed computer systems. *IEEE Network Magazine*, pages 22-28, September 1988.

[5] Sameh Rabie, Andrew Rau-Chaplin, and Taro Shibahara. DAD: A real-time expert system for monitoring of data packet networks. *IEEE Network Magazine*, pages 29-34, September 1988.

[6] Wei-Dong Zhan, Suchai Thanawastien, and Lois M.K. Delcambre. SimNetMan: An expert system for designing rule-based network management systems. *IEEE Network Magazine*, pages 35-42, September 1988.

[7] Mark T. Sutter and Paul E. Zeldin. Designing expert systems for real-time diagnosis of self-correcting networks. *IEEE Network Magazine*, pages 43-51, September 1988.

[8] Robert N. Cronk, Paul H. Callahan, and Lawrence Bernstein. Rule-based expert systems for network management and operations: An introduction. *IEEE Network Magazine*, pages 7-21, September 1988.

[9] Gabriel Jakobson, Robert Weihmayer, and Mark Weissman. A dedicated expert system shell for telecommunication network alarm correlation. *IEEE Network Magazine*, pages 277-288, September 1993.

[10] A. Finkel, K.C. Houck, S.B. Calo, and A.T. Bouloutas. An alarm correlation system for heterogeneous networks. *IEEE Network Magazine*, pages 289-309, September 1993.

[11] James A. Leonard and Mark A. Kramer. Radial basis function networks for classifying process faults. *IEEE Control Systems*, pages 31-38, April 1991.

[12] Larry L. Ball. *Network Management with Smart Systems.* McGraw-Hill, 1994.

[13] James Malcolm and Trish Wooding. IKBS in network management. *Computer Communications*, 13(9):542-546, November 1990.

[14] Marshall T. Rose. *How to manage your network using SNMP: the networking management practicum.* PTR Prentice Hall, 1995.

[15] William Stallings. *Network management.* IEEE Computer Society Press, 1993.

[16] William Stallings. *SNMP, SNMPv2, and CMIP: the practical guide to network-management standards*, Addison-Wesley Pub. Co., 1993.

[17] John Mueller. *The hands-on guide to network management.* Windcrest/McGraw-Hill, 1st. edition, 1993.

[18] Rfc 1382: SNMP MIB extension for the X.25 packet layer.

[19] Rfc 1215: A convention for defining traps for use with the SNMP.

[20] X.25 recommendations.