LEARNING RULES FOR PREVENTING AND DIAGNOSING FAULTS IN LARGE-SCALE DATA COMMUNICATIONS NETWORKS: AN EXPLORATORY STUDY

by

T. Janssen
E. Bloedorn
M. Hieb
R. S. Michalski

Proceedings of the Fourth International Symposium on Artificial Intelligence,
Cancun, Mexico, Nov. 13-15, 1991.

LEARNING RULES FOR PREVENTING AND DIAGNOSING FAULTS IN LARGE-SCALE DATA COMMUNICATION NETWORKS: An Exploratory Study

Terry Janssen, Eric Bloedorn and Michael R. Hieb
Artificial Intelligence Support Center
Computer Sciences Corporation and
Center for Artificial Intelligence
George Mason University

Ryszard S. Michalski
Center for Artificial Intelligence
George Mason University

Abstract

This paper presents an exploratory study on the applicability of machine learning techniques to large-scale data communication networks. Specifically, the study addresses the problems of learning two types of decision rules: diagnostic rules for fault recognition, and prediction rules for fault prevention. The methodology of applying to these problems the AQ15 inductive learning program is described and illustrated by examples. The input to the learning program is in the form of internet events. Experimental results have shown significant potential for use of the proposed methodology in this area.

1. Introduction

Determining rules for diagnosing faults in communications networks is a difficult problem, even for well trained network engineers. Network engineers typically learn rules for fault diagnosis only after experiencing recurrent patterns in network behavior. Symptoms in the form of network event messages and network statistics must be observed just prior to the occurrence of faults and failures, and must be recorded with precision.

The problem of learning such diagnostic rules is compounded in complex internets by the occurrence of many types of symptoms, and the presence of many types of devices from which reports of those symptoms are generated, each of which may use a different communication protocol. An internet usually consists of several networks with heterogeneous architectures (Schartz, 1987). Such a system of diverse components is supposed, however, to be functionally integrated so that a message (in the form of a packet) sent from one point can reach any other point in the network. Different internet architectures have been developed. An example of an internet is presented in Figure 1. The figure shows an architecture of the campus-wide network designed and used by Computer Sciences Corporation at its Virginia Technology Center.

These networks pass very large volumes of messages. A message (packet) is a transmission of information from one point in the network to one or more destinations. Due to their complexity, the diagnosis and resolution of faults in such networks is a major problem. When a network experiences an error, e.g., a message does not successfully reach its destination, an event is created within the network. The number of these fault and anomaly events per month in even a moderately sized network can exceed tens of thousands.

It is difficult for any one person to know all of the types of events that may be experienced within an internet; it is even more difficult for a person to determine all of the plausible causes of such events in the context of specific event situations.

In this context a promising and important idea is to apply modern machine learning to this problem. This paper presents an approach to machine learning of rules from example events that occur within an interneted network operation. The approach is based primarily on the AQ15 machine learning program (Michalski, 1986a). This learning approach is presented in Section 6. Before the approach can be understood, the concepts of faults, failures and errors, and the causes of network malfunctions must be understood. These concepts are described in the following sections.

2. Network Malfunctions

Data communications networks send data between devices in the network by packets. A packet is a sequence of bits, organized into data fields, that contains one or more messages, and a header and trailer of control information. Whenever a malfunction is detected by the system, an *event message* is generated. An event message may indicate a fault, a failure, or an error (FFE) within the network.

There does not exist a common definition for faults, failures and errors as they relate to digital networks. In some cases the terms are synonymous (Cheng, et al, 1970). In other cases, different terms have been used. For example (Pan, 1981) uses failure, degradtion and malfunction in place of our failure, fault and error. For the purpose of this paper, the FFEs are defined as follows. An *error* in a communication network is a single unsuccessful transmission of a message. For example, an error may be caused by electrical interference which momentarily disrupts the message. This type of error can be corrected within the protocol of the network, simply by retransmission of the message. Errors do not usually result in prolonged loss of communication. A *fault* is a more severe error. Severity is defined by error count. If the error count exceeds an error threshold, then a fault has occurred. A fault may be caused, for example, by an overloaded network. The receiving node may be operating normally, but may not be prepared to acknowledge the incoming message due to network traffic. Faults do not usually cause long-term loss of communication; often faults are corrected by network protocol. A *failure* in a data commu-

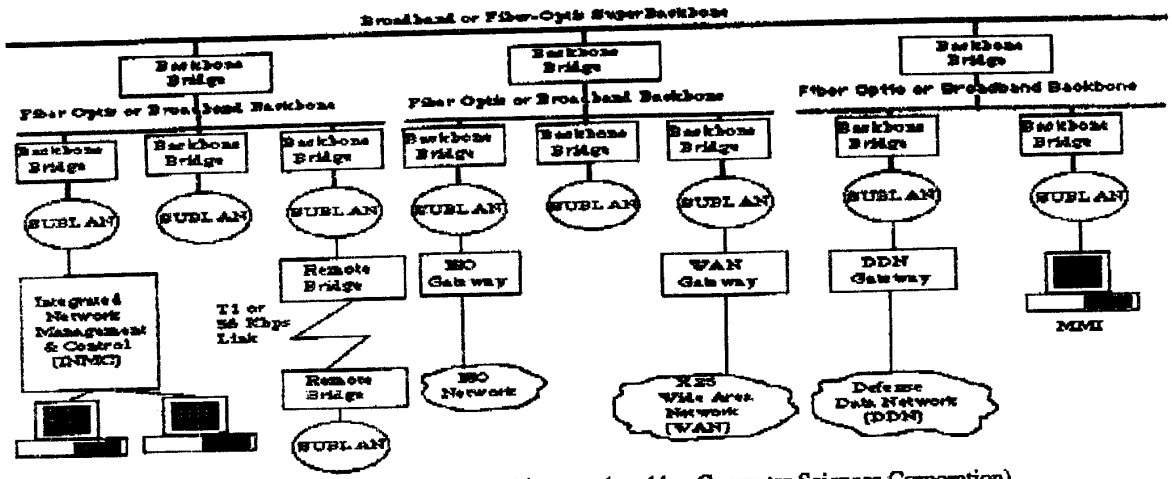

Figure 1. An example of an internet architecture (used by Computer Sciences Corporation).

nications network is defined as the most severe type of malfunction. Failures occur when there exists a complete loss of communication with one or more points in the network. Failures usually cause longer loss of communication. An example of a failure is hardware failure of a specific node in a network. Failures can also be caused by environmental factors. A common example is power failure due to a storm. FFEs usually lead to generation of events messages within a network. An event is generated either as a direct consequence of an observed malfunction or anomaly, or as an indirect consequence, i.e., from a chain reaction.

3. Causes of Network Malfunctions

As indicated above, a network malfunction can be due to a number of causes. Such causes can be internal or external. Internal causes are of two types: hardware or software. Hardware malfunctions are either of electronic or mechanical nature. Software problems occur in one or more software components, each identified as a discrete configuration management item. If an FFE is caused by a software element, it is usually by an error in the software design or in the implementation of the software within specific devices on the network.

Software element causes are usually a result of human error either during software design or implementation, or in configuring the operational software within the internet. An example of the latter is an error in a timing parameter in a communication software element loaded into a device that prevents the normal functions of the communication protocol and disallows expected communication with other devices in a network.

Physical causes must be resolved at the field repairable unit (FRU) level within a particular device within some subnet of the internet. The cause is considered to be field repairable if a field engineer can travel to the location in the network where the cause of the problem exists, and perform some corrective action at that location. Often large data communications networks have a field engineering activity that has the responsibility for repairing problems of this type.

Cause and effect is reported at different levels of abstraction within an internet. The level of abstraction that is most useful for network management is the level necessary to enable the (1) dispatch of network resources, whether human or electronic, to the source of the problem and (2) resolution of the problem. This high abstraction level supports the categorization of events into decision classes, e.g., malfunctions caused by (1) overload, (2) device fault, failure or error, (3) error in network configuration, or (4) transmission failures caused by environmental factors. Figure 2 illustrates types of malfunctions that occur in data communication networks.

There are two types of overload: network overload and device overload. A special type of network overload is channel capacity overload. A special type of device overload is a memory buffer overloading within a specific device.

Device fault, failures and errors are internal and fall into one of the two categories of hardware or software. For the purpose of this paper, a hardware malfunction includes device not ready and device not capable (design inadequacy) states, and physical FFEs. Examples of physical FFEs are error in carrier sense, and device jabber, both problems that occur in networks with local area network architectures. Software malfunctions occur in system and application software, including communications protocol software. A special type of communications protocol software malfunction is a backoff error; a special type of application software malfunction is a database integrity error.

Configuration FFEs include inappropriate design, e.g., incorrect configuration of network devices or inconsistent specification of parameters among devices within the configuration. Address errors are a type of parameter error. Examples of address error sources are gateway routing tables and terminal devices.

Transmission FFEs can be caused by one of the above categories or by environmental causes. Environmental causes include degradation of electrical power quality, environmental interference brought about by physical network configuration, or temperature or humidity levels that exceed equipment tolerance.

4. Network Events

A network event is defined, for the purposes of this paper, as the occurrence of an FFE (discussed previously). A network event message is a partial description of an FFE that has occurred; it describes the event through attributes and values. Some event messages describe the general status of a device or network while others are more specific to an instance of an FFE. Network events can be represented as vectors of attribute values. The specific choice of network attributes depends on network type and the types of information deemed important. Section 6 includes a table of the SNMP variables used in the experiments reported in this paper. The variables are only a fraction of the total number available.

An example of an event message that describes status is a message from a device that reports that the device is *up*, meaning that the device is functioning, or *down*, meaning that it is not. Status messages can be generated by a device itself or by some neighboring device. Some devices are programmed to send a message when they change from one state to another while others need to be queried. Event message generation is usually a function of the network management software and protocols that are used. Modern network management protocols such as the Simple Network Management Protocol provide for both locally generated event messages and device query. In the latter case the network management system asks for information and then generates an event message if specific conditions exist. Both approaches to event message generation provide the same information.

Consider the following example. An event message is generated because a predetermined threshold of *checksum errors* is surpassed. An acceptable number of checksum errors is set by a network engineer and stored in the management information base (MIB). A MIB is a database of information compiled from a data communications network. A MIB typically contains descriptions of all devices within a network (or internet), and operational data specific to each device. Each time a *checksum* error occurs within a device, a counter within the device's MIB is incremented. When the number of checksum errors surpasses the threshold, an event message is generated.

5. Learning Diagnostic Rules from Malfunctions

Network events describe FFE situations that are unique to a particular implementation of a particular network architecture and protocol. One goal of machine learning from internet events is to generalize patterns common to groups of events from within an internet such that the event causes can be more readily diagnosed and rectified. Patterns and correlations can be found between multiple events and data elements (including statistical data) stored within devices in the network. Rules can be formulated from these patterns to abduce cause from an FFE. This research is also focused on discovery of rules that can be stored to predict future network failure from patterns that occur within a network operation prior to network failure. For both diagnosis and prevention, these rules can be stored in a knowledge base available to an expert system. Such an expert system can provide assistance to network operators and engineers in fault prediction, diagnosis and resolution within these large, complex data communications networks (Mathonet, et al 1987).

Learning processes can be categorized into synthetic or analytic types of learning (Michalski, 1989). The categorization is based on the types of inference mechanisms used in the learning process. Synthetic learning is inductive in nature, whereas analytic learning is deductive in nature. Deduction is the process of inferring a fact from general rules and other facts. Induction is the process of inferring hypotheses which explain given facts. Deduction is truth-preserving but induction is falsity preserving, i.e., if any premise was false, the hypotheses generated will also be false.

This research effort uses a synthetic (inductive) learning approach based on a constructive induction technique. The two primary types of synthetic learning are learning from examples and learning by observation.

(1) Learning from Examples: Learning from examples is also called supervised learning. Each example is identified as being either a positive example or a negative example of what is to be learned. Examples can be presented to the machine learning program all at once ("batch learning") or as they occur. The latter form is called incremental learning and is particularly relevant to our problem, as events occur continuously in data communications networks.

AQ has been applied to learning rules for diagnosis of soybean diseases (Michalski, et al 1981). The AQ15 program performs incremental learning of decision rules from examples and is a good candidate for our application.

Another interesting application of learning from examples in a communications network is to learn sequences of patterns that occur over time. The advantage of this approach is that a sequence of network events can be used to predict a future network event. Consider the following situation that was experienced repeatedly in a local area network. A device d1 that connects fiber optic cable of a local area network n1 to a router device d2 was observed burning up. Within a few minutes prior to burnout, other events were observed in the network. First, network traffic increased, followed by increased error rates from d1 and d2. The pattern is summarized as follows, with the ^ symbol representing increase and Malf representing a malfunction:

^(traffic, n1) ^(errors, d1) ^(errors, d2) Malf(burnout, d1)

In future situations involving the same types of network and devices, this sequence of events can be used to predict malfunctions of the network, i.e., burnout. The SPARC machine learning program (Michalski, et al 1987) learns

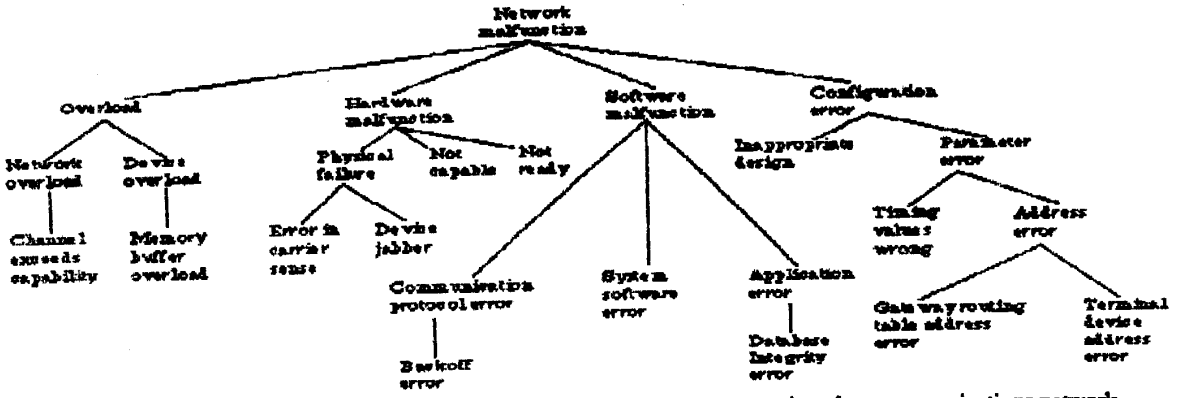

Figure 2. Classification hierarchy of primary malfunctions that can occur in a data communications network.

sequences and is being investigated for learning sequences of network events of this type.

(2) Learning by Observation: Learning by observation is unsupervised. Learning by observation in an internet environment is more limited than learning from examples because individual learning events do not have associated information of cause. The cause is often not determined until several hours after the event has occurred from a problem report submitted by the network operator or field engineer that successfully rectified the problem.

Despite this problem, learning by observation has applicability to internetted networks. The concepts conveyed by an event message may be common among a variety of networks, devices and software programs. Conceptual clustering provides a means of determining commonality in the types of network entities that generate a particular network message type. A classic example of a conceptual clustering program is CLUSTER/2 that has been applied to classification of folk songs from examples (Michalski, et al 1983; Stepp, 1983).

Finally, a machine learning approach also investigated by this research is the discovery of numerical regularities that occur in communications networks prior to network performance degradation or failure. The concept of threshold has been widely applied to communications networks. Thresholds can be used to indicate unacceptable numbers of network errors, and are an important predictor of network performance or network failure. For example, in a local area network, excessive network traffic can cause network performance degradation. Discovery of numerical regularities from network statistics such as traffic and error counts is an import application area. A machine learning program that has been successful in discovery of numerical regularity is ABACUS (Falkenhainer, et al 1986). ABACUS rediscovered Ohm's Law, Stoke's Law, the Law of Conservation of Energy and Kepler's Law, and analyzed chemical compound data. ABACUS is a candidate program for discovery of numerical regularities in MIB data collected from communications networks.

6. Method using the AQ Algorithm

Figure 3 illustrates an approach (based on the AQ15 learning program) to machine learning using network event messages, MIB data and problem reports. Learning takes place from network event data that is collected directly from the network and from information collected by field engineers and network operators. In most cases a network operator observes a network event, diagnoses the cause and creates a problem report to document the event and cause. In cases where field repair is required, a field engineer rectifies the cause and documents it in a problem report. The sequence of three major steps is summarized below.

(1) The first step of this approach is to collect network event information. Collection results in the description of an example of a network malfunction and the network symptoms prior to or shortly after the malfunction. Internet event attributes include a name and a description of the event, the event source, network and device states, and an abstract classification of the type of network malfunction. Event information is derived from a network management system and problem reports created by network operators and field engineers. The MIB is a database of network states and statistics collected by network software at predetermined intervals of time. The problem reports provide additional attributes collected by network operators and engineers that further characterize the event and cause.

(2) The second step is to represent the event description as a sequence of attributes and values. The problem report is parsed for attributes and values that further characterize the event, and the event is assigned to a decision class of cause, i.e., type of network malfunction. The characteristic description describes where the message originated, a description of the event as conveyed by the internet message, the object of the message, and properties of the network at the time of event occurrence.

(3) The third step is to generate rules from the event descriptions using the AQ15 inductive learning program. The goal of this learning is to generate rules capable of diagnosing or predicting network faults.

Network events are categorized into decision classes, i.e., the causes of the events, prior to learning. Each network event is a member of some decision class, i.e., a positive

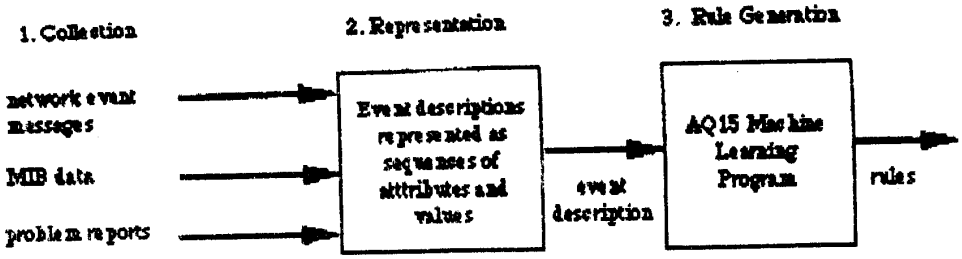

Figure 3. Illustration of method for the application of machine learning to network management.

example of that decision class. A negative example is one that does not belong to the decision class. The AQ15 inductive learning program generalizes decision rules by covering all positive examples within a decision class but none of the negative examples.

7. A Brief Review of the AQ Algorithm

Because the AQ algorithm is used as an important module of this method, for completeness we provide a brief description of it. The AQ algorithm generates the minimum or near-minimum number of general decision rules characterizing a set of instances, as described in (Michalski, 1986a).

(1) A single positive example, called a seed, is selected and a set of most general conjunctive descriptions of this example is computed (such a set is called a star for the seed). Each of these descriptions must exclude all negative examples.

(2) Using a description preference criterion, a single description is selected from the star, called the "best" description. If this description covers all positive examples, then the algorithm stops.

(3) Otherwise a new seed is selected from among the unexplained (uncovered) examples, and steps 1 and 2 are repeated until all examples are covered.

The disjunction of the descriptions selected in each step constitutes a complete, consistent and general description of all examples. The preference criterion used in selecting a description from a star is expressed as a list of elementary criteria that are applied lexiographically and with a certain tolerance. The criteria may be simplicity of description (measured by the number of variables used), cost (the sum of the given costs of the individual variables), or other criteria.

The description of a class is expressed using the variable-valued logic system 1 (VL_1), which is a multiple-valued logic propositional calculus with typed variables (Michalski, 1974). A class description is called a cover. A cover of a concept is a disjunction of complexes describing all positive examples and none of the negative examples. A complex is a conjunction of selectors, which is the simplest statement in VL_1. A selector relates a variable to a value or a disjunction of values, for example [temperature = cold, warm], or [x < 5]. The general form of a selector is:

$$[L \# R]$$

where L, called the referee, is an attribute, and R, called the referent, is a set of values in the domain of the attribute in L, and # is a relational symbol which can be one of the following: =, <,>,>=,<=,<>.

8. Experiments and Results

In an attempt to judge the applicability of machine learning to network fault management, several preliminary experiments were devised. The data for these experiments was derived from a simulated network. The first step in designing this experiment was determining the general characteristics of the attributes used to describe the solution space. The network domain provides a large amount of data in myriad formats. Both data and format are dependent on the network protocol used.

For this network the SNMP protocol was chosen and a subset of the data elements provided by the SNMP MIB was selected to be utilized in our experiments. These SNMP data elements were the attributes that described a network event to our learning program. Each data element could assume a value and was considered as a variable. Most of the data elements were counters.

The SNMP variables are presented in Table 1. Each variable's name and a brief description is given, along with the set of values that it could assume. Only one variable had nominal values, and the other nine had real or continuous values, with different ranges. These ranges were created for the experiments.

A network configuration was designed that was complex enough to be a valid test of our method, while at the same time conceptually simple enough to verify the results. The network consists of 5 nodes as shown in Figure 4. Node 5 (N5) is the SNMP Network Management Station, which queries the other nodes' MIB. Given this network model we determined what network event messages the network operator would receive from the Network Management Station if certain faults or failures occurred in the network. An event was simply a characterization of the state of a node in our network, using the SNMP variables as attributes.

For example, if Node 1 (N1) went down, the operator need only to check the ifOperStatus parameter for that node. In most cases however, the description of network status during or after a fault or failure has occurred is more complicated. Four faults/failures were postulated. The faults were either a noisy line or an intruder breaking into the network. The failures were either a modem down or the interface to the node down.

A number of errors were generated on the network and corresponding network event messages were created. These events consisted of attribute vectors which were assigned to one of the faults/failures indicated above or to a normal class. This yielded a total of 5 classes. These preclassified events were given to AQ15 in its batch learning mode

Initial results were poor because the generated rules consisted of complicated disjunctions of values. For example, rules of the following form were produced [ifInDiscards = 308, 896]. A more useful rule would be [ifInDiscards=high]. In order to achieve this generalization of values we initially hand-scaled the data. With these new values better rules were produced. Automatic scaling was then implemented. This is a simple scaling which partitions the total range into equal sub-ranges. The calculated scaled value replaced the original value and the AQ algorithm proceeded to construct discriminant descriptions of the error classes as it normally would.

In the first experiment the learning set consisted of 5 classes (4 error-status classes and 1 normal-status class) described by 5 attributes (the first 5 listed in Table 1). There were between 2 and 4 events per class for the 4 error classes and 20 events in the normal class. The rules produced from this learning data are shown below. The values in experiment 1 were partitioned into 4 levels: very_low, low, high and very_high.

```
Modem Down :: > [ifOperStaus=up]
                [ipInAddrErrors = very_high]
Noisy Line  :: > [ifInerrors = high ..very_high]
Node Down   :: > [ifOperStatus = down]
Intruder    :: > [ipInHdrErrors = low, very_high]
Normal Status :: > [ifInDiscards = very_low..low]
                   [ifInErrors = very_low]
                   [ipInaddrErrors = very_low ..low] or
                   [ifInDiscards = very_low]
                   [ifInErrors = very_low..low]
```

In the second experiment the learning set consisted of the same 5 classes, but this time there were 10 attributes (those listed in Table 1). Once again there were between 2 and 4 events per class for the error classes and 20 events for the normal class. The rules produced from this learning data are shown in Figure 6. The values in experiment 2 were partitioned into 6 levels: ext_low, very_low, low, high, very_high, ext_high.

```
Modem Down :: > [ifInErrors = very_low]
Noisy Line :: > [ifReasmFails=low,
                very_high..ext_high]
                [icmpInErrors = high..ext_high]
```

Name	Description	Values
ifOperStatus	Current operational state of the interface	up,down
ifInDiscards	Number of inbound packetswhich were chosen to be discarded.	0-1000
ifInErrors	Number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol	0-100
ipInHdrErrors	Number of input datagrams discarded due to errors in their Internet Protocol headers.	0-2000
ipInAddrErrors	Number of input datagrams discarded because the Internet Protocol address in their header's destination field was not a valid address to be received at this entity	0-1000
ipForwDatagrams	Number of input datagrams for which this entity was not their final Internet Protocol destination, and an attempt was made to find a route to forward	0-2000
ipOutNoRoutes	Number of Internet Protocol datagrams discarded because no route could be found to transmit them to their destination.	0-250
ipReasmFails	Number of failures detected by the Internet Protocol re-assembly algorithm.	0-400
icmpInErrors	Number of Internet Control Message Protocol messages which the entity received but determined as having errors.	0-300
icmpInTimeExcds	Number of Internet Control Message Protocol Time Exceeded messages received.	0-500

Table 1. SNMP variables used in experiments.

```
Node Down   :: > [ifOperStatus = down]
Intruder    :: > [ipInHdrErrors = low, ext_high]
Normal Status :: > [ipInaddrErrors = ext_low]
                   [icmpInErrors = ext_low..low]
                   [icmpInTimeExcds=ext_low..high]
```

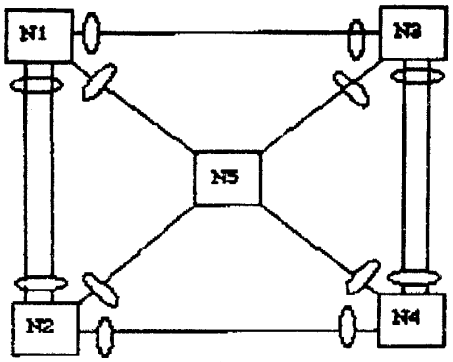

Figure 4. Schematic of network used for experiments

9. Discussion of Results

The results demonstrate that characteristic descriptions of network events can be used to generate rules for fault diagnosis, i.e., abduction of cause from network event descriptions. The experiments demonstrate machine learning from an internet operation. The experimental data used for the experiments produces interesting rules. The learned rules are not trivial and not intuitively obvious. Furthermore, the machine learning program generated the rules within seconds, much faster than a person can generalize rules from the same data.

Learning from examples, whether by human or machine, is prone to errors from two sources. The first is from learning with incomplete, imprecise or incorrect information. The second source of error is the generalization process. Errors of the second kind can be avoided by using a well-founded machine learning program. Errors of the first kind are expected and tolerable because only a small portion of the learning space is covered by the example events. A complete event space is one which includes examples of all possible events. This is an unrealistic condition in real domains.

The rules produced by the machine learning program are general and can be tested against new events produced by the internet. Because generalization involves dropping conditions, the rules are not always correct or complete for future events. It is expected that the attributes available in the protocol are sufficiently precise to discriminate different faults, yet produce useful rules for recognizing new events.

10. Conclusion and New Research Topics

The proposed method of applying machine learning to create descriptions of network malfunctions performed well in experiments. Clear, concise descriptions were quickly calculated for a number of network states including: modem-down, noisy-line, node-down, intruder and normal-status. These descriptions were derived from vectors of SNMP variables which characterized the state of the network. In the two experiments reported in this paper, only a fraction of the available SNMP variables and possible network error states were considered. Further experiments will increase both the complexity of the network from which the experimental data are derived and the number of SNMP variables used in the description space.

There are numerous extensions to this research to make it more efficient and practical. The most promising approaches are 1) to use constructive induction to produce more concise network descriptions, 2) to utilize positional information for a better representation and 3) to utilize temporal information for a better representation.

Constructive induction could discover concepts that are not contained within event examples. Concepts such as thresholds or ratios of network attributes are very useful for diagnosing specific causes of faults. These concepts are critical to characterizing symptoms that occur before network degradation or failure. These concepts can also be used to generalize rules for use in a broad number of similar, but not identical, FFE situations caused by the same type of malfunction within the network.

An example of a useful ratio is rho. Rho is a measure of network throughput and is computed by dividing network utilization by network capacity. When rho exceeds an acceptable threshold level, network performance begins to degrade. Rho also represents the percentage of network capacity that can be utilized before traffic congestion will contribute to a communication failure. Given the concept of rho, the following could be induced from the network event information:

If the network type is *local area network*, and
the event type is *direct memory overrun*, and
the threshold of rho is *exceeded*,
Then a plausible cause is *device overload*.

The concept of rho or some other relationship of network attributes can be derived using constructive induction. In one method currently being researched, the rule learning program may be instructed to discover 'useful' ratios of given attributes. The program uses a generate-and-test method to discover these relationships. In another method, ratios that are thought to be useful but are not present in the original data can be explicitly defined for the program. The program calculates the value of this ratio for all the data automatically.

The second extension to the current research involves using positional information. Currently, positional information is not included. There are some network faults that cannot be captured by such a simple representation. For example, an increase in traffic to a particular node may be due to the failure of nearby nodes. The AQ learning program currently being used cannot use multiple-place predicates such as *connected-to*, which are suitable for capturing positional information. The INDUCE program (Michalski and Stepp, 1986b) is able to use such predicates and in fact has been used to derive general rules for the structure of chemical compounds from examples. These chemical structures are very similar in representation to network graphs.

The third extension to this work involves reasoning about

time. As in the case of positional information, temporal information, is not included in the data. We propose to tag each dynamic attribute value with a time stamp. Given this information useful derived attributes such as rates in error counts could be identified. In addition, some temporal ordering of events can be done such as "event x before event y". Once these trends are identified they may become a part of even more elaborate diagnosis or predictive rules.

Finally, we plan to test this approach on a large number of FFE event examples. This large number of examples is necessary to determine the precision and correctness of this method for a wide variety of internet states. The goal of this research is to develop a method sophisticated and robust enough so that an engineered and fielded machine learning system can be implemented as part of a data communications network.

Acknowledgements

The authors wish to thank Kenneth Kaufman and Gheorghe Tecuci for their helpful comments and suggestions.

This research was supported by DARPA grants administered by ONR, N00014-87-K-0874, and N00014-91-J-1854, and in part by the Office of Naval Research under grants N00014-88-K-0226, N00014-88-K-0226, N00014-88-K-0397 and N00014-91-J-1351, and Computer Sciences Corporation.

References

Falkenheiner, B. and Michalski, R. "Integrating Quantitative and Qualitative Discovery: The ABACUS system," ISG-86-17, UIUCSDCS-F-86-967, University of Illinois, Urbana, May 1986.

Cheng, H.Y., Manning, E., and Metze, G., *Fault Diagnosis of Digital Systems*, Wiley-Interscience, New York, 1970.

Hong, J., Mozetic, I., and Michalski, R.S., "AQ15: Incremental Learning of Attribute-Based Descriptions from Examples, the Method and User's Guide," ISG 86-5, UIUCDCS-F-86-949, Department of Computer Science, University of Illinois, Urbana, May 1986.

Mathonet, R., and Cotthem, H., and Vanryckeghem, L. "DANTES: An Expert System for Real-Time Network Troubleshooting", Tenth International Conference on Artificial Intelligence, Milan, 1987.

Michalski, R. and Chilausky, R. "Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from Examples: A Case Study Involving Soybean Pathology," in a chapter of *Fuzzy Reasoning and its Applications*, 1981.

Michalski, R.S., Carbonnel J., and Mitchell T., (Eds.), *Machine Learning: An Artificial Intelligence Approach* Vol I, TIOGA Publishing Co., Palo Alto, 1983.

Michalski, R.S., Mozetic, I., Hong, J., and Lavrac, N., "The AQ15 Inductive Learning System: An Overview and Experiments," ISG 86-20, UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois, Urbana, 1986a.

Michalski, R., and Stepp R. "INDUCE 3: A Program for Learning Structural Descriptions from Examples," ISG 86-9, UIUCSDCS-F-86-960, Department of Computer Science, University of Illinois, Urbana, 1986b.

Michalski, R.S., Ko, H., and Chen, K. "Qualitative Prediction: The SPARC/G Methodology for Inductively Describing and Predicting Discrete Processes," chapter in Current Issues in Expert Systems, edited A. Van Lamsweerde and P. DuFour, 1987.

Pau, L.E., *Failure Diagnosis and Performance Monitoring*, Control and Systems theory, vol. 11. Marcel Dekker Inc. New York, 1981.

Schartz, M. *Telecommunications Networks: Protocols, Modeling and Analysis*, Addison-Wesley, Reading, MA, 1987.

Stepp, R. "A Description and User's Guide for CLUSTER/2, A Program for Conjunctive Conceptual Clustering," Report No., Department of Computer Science, University of Illinois Urbana, November, 1983.