Fault Identification in Computer Networks: A Review and a New Approach

Tim Oates

Computer Science Technical Report 95-113

Experimental Knowledge Systems Laboratory
Computer Science Department, LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610

Abstract

As computer networks increase in size, heterogeneity, complexity and pervasiveness, effective management of such networks simultaneously becomes more important and more difficult. This paper explores in detail one aspect of network management, *fault identification*. Fault identification is the process whereby the existence and nature of network faults are ascertained. Characteristics of the fault identification problem are explored and existing approaches are surveyed. Interestingly, much of the work in this area makes use of techniques from Artificial Intelligence, especially expert systems. Features of the fault identification problem that make it is amenable to AI approaches and resistant to more traditional algorithmic solutions are examined. Finally, a new approach to fault identification is proposed that employs an algorithm for finding dependencies among values in multiple streams of data over time. The new approach is compared to existing approaches, and it advantages and disadvantages are weighed.

1 Introduction

As businesses and individuals have become increasingly reliant on computer networks, the complexity of those networks has grown along a number of dimensions. The phenomenal growth of the Internet in recent years provides a clear example of the extent to which the use of computer networks is becoming ubiquitous [1]. In July of 1993, there were 1.8 million hosts on the Internet. That number rose to 3.2 million in July of 1994, and climbed to 6.6 million in July of 1995. It is estimated that 70,000 networks comprise the Internet, and that there are between 20 and 40 million users worldwide. As the demands that users place on networks become more complex and varied, so do the networks themselves. Heterogeneity is the rule rather than the exception. Data from a variety of applications may travel under the control of different protocols through numerous physical devices and transmission media. Large numbers of software and hardware vendors sell network components that have varying capabilities and operating characteristics. Network configurations and demand for network resources are highly dynamic. There is general consensus that this trend toward increasing complexity will continue rather than abate.

As computer networks grow in complexity and pervasiveness, effective management of those networks simultaneously becomes more important and more difficult. Users of network aware applications are not concerned with the welter of complexity that lies on the other side of their ethernet card, they simply expect their application, and thus the network, to provide service at a given level of quality. For example, a specialist at a large hospital may use a remote X-ray application to consult on cases where such expertise is not available locally. Regardless of the state of the network, the specialist expects X-ray images to arrive quickly, securely, and without errors. Likewise, corporations expect mission critical applications to perform their functions in spite of problems in the network, whatever their location and nature. Unfortunately, maintaining smooth operation of modern, complex networks is a difficult task. The amount of information that can be gathered about the state of a network for the purpose of making management decision is staggering. Different components (both software and hardware) from different manufacturers can fail for different reasons, and the manifestations of such failures can be diverse both topologically and evidentially. In addition, as larger volumes of data move more and more rapidly over high bandwidth media under the control of high speed protocols, the rapidity with which management decisions are made becomes crucial.

The network management rubric encompasses a large number of tasks, and various standards bodies are attempting to bring order and organization to those tasks. The International Standards Organization (ISO) has divided network management tasks into six categories as part of their Open Systems Interconnection (OSI) model: configuration management, fault management, performance management, security management, accounting management and directory management. One of the goals of categorizing network management tasks is to facilitate modularization in the design and implementation of network management tools. As we will see, network management tasks often do not fall neatly into just one of the six categories. For example, the only manifestation of a network fault may be degraded performance. In addition to clearly defining the network management problem, various ISO and Internet standards have been proposed for solutions to certain aspects of the problem. For example, the Simple Network Management Protocol (SNMP) is an Internet standard protocol for communication between managing and managed entities. The analogous ISO standard is the Common Management Information Protocol (CMIP). There are also Internet and OSI standards for the structure and content of information about managed entities (i.e. the state of the network).

In this paper we are concerned with one aspect of fault management, fault identification. Our goal is to clearly define the fault identification problem, explore issues related to that problem, and review existing approaches that have been discussed in the network management literature. The *fault management* task can be characterized as detecting when network behavior deviates from normal and formulating a corrective course of action. Fault management can be decomposed into three subtasks: fault identification, fault diagnosis, and fault remediation. The *fault identification* subtask involves detecting that a problematic deviation from normal behavior has occurred and identifying its nature. Fault diagnosis involves determining the root cause of an identified problem, and fault remediation is the formulation of a course of action that will repair the problem. All three stages of fault management involve reasoning and decision making based on information about current and past states of the network. We will see that two dimensions along which approaches to fault management (and fault identification) vary are the nature of the information used to make those decisions and the degree to which decision making is automated.

Interestingly, much of the work in this area makes use of techniques from Artificial Intelligence (AI), especially expert systems and, increasingly, machine learning. The complexity of computer networks and the time critical nature of management decisions make network management a domain that is difficult for humans. Expert systems have achieved performance equivalent to that of human experts in a number of domains, including certain aspects of network management [3]. However, the dynamic nature of computer networks often makes the knowledge contained in expert systems *brittle*. That is, such knowledge quickly becomes out of date and ineffective as network topology, composition and usage change over time. Therefore, knowledge acquisition from human experts, a notoriously difficult task, often becomes a bottleneck. Machine learning techniques are gaining favor in the network management community as one way around the *knowledge acquisition bottleneck* by automating the acquisition of knowledge about dynamically changing network environments [10].

After defining the fault identification problem and reviewing existing approaches, we present a new approach based on a machine learning algorithm, called Multi-Stream Dependency Detection (MSDD), that learns dependencies among values in multiple streams of data over time [26]. MSDD accepts as input a history of vectors of categorical values that describe the state of the network as it changes over time. From that history, MSDD produces a set of probabilistic rules that describe how various features of network state are predictive of future states of the network. In particular, if the state vector contains information about faults that have occurred in the past, MSDD can learn rules that probabilistically relate current network state to future faults. Such rules facilitate proactive network management, alerting network managers to the possibility of trouble in the future. In addition, the rules learned by MSDD constitute knowledge about the dynamics of the network. MSDD can be run periodically as the dynamics of the network change, automatically acquiring up-to-date knowledge and thereby avoiding brittleness.

The remainder of this paper is organized as follows. In Section 2 we define the fault identification problem somewhat more formally and examine some of the issues that one must consider when formulating solutions to that problem. Section 3 contains a discussion of several industry standards that are important and relevant to network management in general and fault identification in particular. Those standards are the foundation upon which most solutions to network management are built, so a thorough understanding of their scope and function is important for the review in Section 4 of approaches to fault identification that have appeared in the network management literature. Our review of existing approaches is followed in Section 5 by a proposal for a new approach to fault identification based on the MSDD algorithm. Our approach automates the acquisition of knowledge that enables proactive fault identification. Finally, we summarize and conclude in Section 6.

2 The Fault Identification Problem

We have previously defined fault identification as the process whereby the existence and nature of network faults are ascertained. A *fault* is simply a malfunction in some component of the network, either hardware or software. For example, a parity error in a router's internal memory that causes it to crash, or a software bug in some portion of a protocol stack that causes anomalous behavior. At an abstract level, fault identification can be thought of as a function, $\mathcal{I}$, with inputs and outputs. The input to our function is a description of network state, $\mathcal{S}$, and the output is a set of hypotheses, $\mathcal{H}$, concerning the existence and nature of network faults. The state $\mathcal{S}$ is meant to convey, in some sense, the health of the network. The hypothesis set $\mathcal{H}$ may be empty, indicating that no faults are believed to be present in the network. Alternatively, $\mathcal{H}$ may contain $n \geq 1$ hypotheses concerning the existence of n distinct faults. Each hypothesis may specify the indications in $\mathcal{S}$ of the corresponding fault and may contain some amount of diagnostic information. That is, identification and diagnosis are rarely totally decoupled. Fault identification, therefore, is a process or function that maps from network states to fault hypotheses: $\mathcal{I}(\mathcal{S}) = \mathcal{H}$.

Our characterization of the fault identification problem points to at least two dimensions along which approaches to the problem vary. First, different approaches to fault identification define $\mathcal{S}$ differently. At the lowest level, information about the state of individual network components (e.g. routers, bridges, ethernet segments, token ring cards) can be recorded over time. Examples of that type of information include queue lengths, buffer usage, throughput, and retransmission rates. This type of data, often called *sensor data*, serves as the lowest level input to three of the six OSI network management categories: fault management, performance management and accounting management. The amount of sensor data available in large networks can be as high as $20 - 30$ gigabytes per day [12]. Clearly, some amount of abstraction and/or summarization of such vast quantities of raw data is desirable. One common approach is to maintain summary statistics for sensor data, such as means or cumulative values, rather than time series [33]. Another is to establish thresholds on the values of state variables for individual components, and to restrict $\mathcal{S}$ to contain information about violations of those thresholds [6, 19]. Threshold violations are often called *events* or *alarms*. To facilitate human reasoning about network state, sensor data is often presented graphically, at a very high level of abstraction. For example, links may be represented as pipes of varying diameter to indicate the amount of traffic they carry [4].

The second dimension along which approaches to fault identification often vary is the degree to which the computation of $\mathcal{I}$ is automated. At one extreme, human users are solely responsible for ascertaining the existence of faults. Tools may be available to examine state information at different levels of detail, but no tools exist to automate reasoning about the presence of faults. At the other extreme, state information is fed directly to an automated process that emits $\mathcal{H}$ with no human intervention. Most existing approaches are mixed initiative, lying somewhere on the spectrum between the extremes (although the level of automation is typically very low).

As networks become more complex and management decisions become more time critical, there is pressure to move toward more compact, abstract representations of $\mathcal{S}$ for human consumption and to higher levels of automation. That is, humans can process only a tiny fraction of the raw data that is potentially available, and they can do so at speeds that may not be sufficient for today's high capacity networks. When gigabits of data per second flow through a network, every second that the network operates at suboptimal levels can be very costly. That same pressure is also causing the move toward increasing use of AI techniques. Automated reasoners have the potential to deal with more information more rapidly than human reasoners, thereby alleviated some of the burden on human network managers and improving the overall quality of network operations.

Why is fault identification a difficult task? Consider a simple example in which a link in an internetwork fails. That one failure can have effects that ripple throughout the internetwork: applications may time out on requests that traverse the link; errors may be reported as new connections to hosts on the link fail to be established; as routing tables get automatically updated and network traffic shifts away from the failed link, other links may become overloaded and begin to report performance degradations. The result is that a wide variety of errors from geographically dispersed locations may suddenly flood the network managers console. The manager is left with the difficult task of determining if the problem is temporary or persistent, if multiple problems are occurring simultaneously or if one problem is having widespread effects, and exactly where in the vast internetwork the problem exists. That task is made still more complex by the fact that different faults may have very similar manifestations. Suppose a host, rather than a link, had failed in our example above. We would still expect to see applications timing out and new connections failing to be established. However, those errors would be restricted to the one host rather than all hosts on the link. The result is that two very different problems may have remarkably similar symptoms.

The distributed nature of computer networks also contributes to the difficulty of fault identification. Due to nondeterministic message delays, the times at which the effects of faults are felt in various parts of the network are highly variable. Therefore, secondary and tertiary effects of a fault may depend on the order in which its primary effects manifest themselves. Likewise, reports of the effects of a fault may arrive at the manager's station in an almost arbitrary order, irrespective of the order imposed by time of occurrence and causality. Although such reports could be timestamped, that would require clock synchronization throughout the network. Finally, a network experiencing a fault may be under stress, leading to packet loss and potentially loss of information relevant to identifying the nature of the fault. The end result is that identical faults can have vastly different manifestations, making the reverse mapping from manifestations to faults very hard to compute.

3 Network Management Standards

In the interest of obtaining truly integrated (vendor-independent, interoperable) network management, various organizations have attempted to standardize aspects of the network management problem. In particular, much attention has been focused on the structure, content, and manipulation of network state information. If the syntax and semantics of sensor data obtained from each network device were dependent on the manufacturer of that device, synthesizing a coherent, accurate view of the overall health of the network would be a horribly complex task. Rather, various standards specify the structure and content of management information that must be maintained for specific network devices in a vendor independent manner. Coupled with standards for accessing and manipulating that information, vendors and researchers can focus on the more important problem of how to make management decisions based on (standardized) state information.

There are basically two models of network management, each with its own set of standards, that are vying for market dominance. The *Internet model* is a minimalist approach, both in terms of functionality and the requirements that it places on network components, that is targeted at management of TCP/IP based internetworks (e.g. the Internet). The *OSI model* is more powerful and general than the Internet model, for which it sacrifices simplicity. The Internet model's minimalism has made it attractive to those interested in actually implementing network management software, and it has therefore achieved market dominance. The Internet and OSI camps are strongly divided, though, on the question of whether the Internet model can be extended to non-TCP/IP environments. That is, is the Internet model the basis for a solution to the general network management problem, or is a move to a more OSI-like model inevitable?

Most research in network management (including fault management and fault identification) assumes one of the models described above. Therefore, a thorough understanding of both models is important background for the review of previous work in fault identification that follows this section.

3.1 The Internet Model

The Internet model divides network components into two classes: *network management stations* (NMSs) and *network elements*. Network management stations are hosts running management applications that monitor and control network elements. Network elements are devices of any sort that can be managed, such as hosts, routers, bridges, ethernet cards, etc. The software residing in network elements that processes and responds to management commands from network management stations is called an *agent*. One goal of this architecture is to minimize overhead at the network elements. If network management is to be ubiquitous, then all network devices must be manageable. However, if supporting network management functionality severely hinders the performance of routers, bridges, modems and other devices, then manufacturers may not choose to implement that functionality. Therefore, the functionality and computation required at the agents is strictly limited. Rather, it is assumed that specific hosts throughout the network (the NMSs) will be dedicated to performing sophisticated network management functions, and the bulk of the computational burden should be placed there.

The Internet approach views managed objects as collections of scalar variables residing in a virtual store, and models all management functions as setting and getting the values of those variables. Objects are defined according to the Internet standard *Structure of Management Information* [30] which specifies rules for naming objects, a syntax for describing the structure of objects, and rules for encoding objects for the purpose of communicating them. In particular, objects are described with a subset of Abstract Syntax Notation 1 (ASN.1) [27] and are encoded with the Basic Encoding Rules (BER) [28].

The actual collection of objects that can be managed and the operations that can be performed on those objects is called the Management Information Base (MIB). The first Internet standard MIB [29] defined 111 manageable objects for the Internet protocol suite. Vendors and researchers were free to extend this core set of objects within the rules of the SMI standard via an "experimental" subtree of objects and a similar "enterprises" subtree. The current Internet standard MIB, MIB-II [21], contains 170 manageable objects.

To make the ideas contained in the previous paragraphs more concrete, consider the following object defined for the Internet standard MIB-II that indicates the amount of time in hundredths of a second since the network management portion of the system was last reset:

```
sysUpTime OBJECT-TYPE
   SYNTAX  TimeTicks
   ACCESS  read-only
   STATUS  mandatory
   ::= { system 3 }
```

The syntax of the definition above (not to be confused with the SYNTAX portion of the definition itself) exercises a subset of ASN.1 as specified by the SMI. In addition, the SMI requires that each object have a name (**sysUpTime**) and a syntax. In this case, the syntax for the object indicates that it comprises a single scalar value of type **TimeTicks**. That value can only be read by the NMS, as indicated by the ACCESS field, and it must be supported by all MIB-II implementations, as indicated by the STATUS field. The SMI also designates the use of the BER to encode the contents of this and all other objects into bit streams for transmission. The specification of the above object and 169 others constitute the contents of the Internet standard MIB-II.

Network management stations and network elements communicate via the Internet standard Simple Network Management Protocol (SNMP) [2]. SNMP operates on top of the User Datagram Protocol (UDP), a connectionless protocol that does not guarantee datagram delivery. Network management services are often most crucial during times of network stress, and the designers of SNMP felt that the use of connection oriented protocols (e.g. TCP/IP) might add to network stress or simply be infeasible in such situations. In keeping with the model of network management as the getting and setting of variables, SNMP supports four basic operations: **Set, Get, GetNext** and **Trap**. The **Set** and **Get** operations allow an NMS to modify and obtain the value of a MIB variable at a network element. For example, the NMS could instruct an agent to perform a specific management operation by setting the value of a particular variable. **GetNext** allows an NMS to iteratively traverse MIB variables. Note that the **Set, Get** and **GetNext** operations implement a request/response protocol, with the managing entity driving the entire process. The **Trap** operation, however, allows the agent to report events, such as threshold violations, asynchronously to the NMS.

3.2 The OSI Model

In the OSI model, managers and agents use a reliable transport to communicate over associations between application layer processes. Note that this requires all network components involved in management, managers and agents, to support all seven layers of the protocol stack. The management protocol is the OSI standard Common Management Information Protocol (CMIP) [13] which is supported by the OSI standard Common Management Information Service (CMIS) [14]. Just as agents in the Internet model control MIBs, so do agents in the OSI model. However, the structure of the OSI MIB is object oriented, and is therefore very different from the Internet model of objects as collections of scalar variables. OSI managed objects are defined by their attributes, behaviors, operations and notifications. Attributes are data maintained by the object, such as sensor data. Operations are actions that can be taken on the object, and behaviors specify changes that take place within the object in response to operations. Notifications are sent asynchronously by the object to report events.

The operations provided by CMIS are the following: **M-GET, M-CANCEL-GET, M-SET, M-ACTION, M-CREATE, M-DELETE** and **M-EVENT-REPORT**. **M-GET** and **M-SET** allow the manager to obtain and change the value of an object's attribute. **M-CANCEL-GET** cancels a previously issued **M-GET**. **M-ACTION** allows the manager to take object specific actions from a set that has been agreed upon by the manager and the agent. **M-CREATE** and **M-DELETE** allow the manager to request that the agent create and delete instances of managed objects. **M-EVENT-REPORT** allows the agent to report events to the manager asynchronously.

3.3 Discussion

It is important to understand the scope of these standards. They simply provide a vendor-independent mechanism for specifying and manipulating the contents of management information bases. That is, they provide vendor-independent access to network state information. Therefore, researches and vendors who design network management tools within either the Internet or OSI frameworks can expect their tools to interoperate with other tools and devices that operate within that same framework.

However, there are a host of important and difficult questions that these standards were not designed to address. What state information is most relevant to the task of fault identification? How does one decide based on state information whether a fault exists, how many faults exist, where a fault has occurred, and whether the problem is temporary or persistent? What level of summarization and abstraction of the raw sensor data available in MIBs is most appropriate? Should management decisions be made locally with local information by specific components of the network, or should they be made globally with global information at the network level? In the sections that follow, we turn our attention to some existing answers to those questions.

4 Current Approaches to Fault Identification

In this section we survey some current approaches to the fault identification problem. The majority of fault management systems in use today involve very little automated reasoning. Typically, network managers have access to application software that allows them to browse network state at varying levels of detail, and they must rely on their own domain knowledge to ascertain the existence and nature of faults. Therefore, our survey begins in Section 4.1 by looking at common approaches and issues related to network monitoring, the collection and presentation of information about network state. As the need for automated network management grows more pressing, the amount of research in that area increases, but few techniques have gained widespread acceptance. The most common approach to automating part of the fault identification problem, called *event correlation*, is discussed in Section 4.2. That is followed by a broad survey of a number of additional approaches to automating knowledge acquisition and reasoning for the purpose of fault identification and management in Section 4.3. While many of the approaches described in that section show promise, few have been shown to scale well (expert systems being a notable exception) and have therefore not been widely implemented.

4.1 Network Monitoring

Network monitoring is the process of gathering information about the state of a network, synthesizing it into a coherent view of the network's health, and presenting that view to a human network manager in an intuitive and understandable manner. Unfortunately, monitoring of computer networks is often a difficult task. Monitored devices may be widely distributed geographically and may be connected by communication links of varying capacity. The result is that events from those devices may arrive at management stations in arbitrary order, irrespective of orderings imposed by time and causality. Identical faults occurring on subsequent days may generate very different sequences of events. In addition, networks under stress, where event monitoring and network management are most important, may lose packets and thus lose valuable information about relevant events. Finally, the sheer volume of information that is available to be monitored in large networks leads to problems in processing it quickly enough and presenting it effectively.

Network monitoring can be divided into two types: time-driven monitoring and event-driven monitoring. The former involves periodically obtaining snapshots of network state, e.g. by polling for the values of MIB variables. The former involves asynchronous receipt of notifications about "interesting" events. Of the two approaches, event-driven monitoring is by far the most common. Given a good definition of what makes events interesting, event-driven monitoring substantially reduces the amount of network state information that must be considered. Events can be generated by managed objects themselves, or they can be generated by external processes that monitor the status of objects (perhaps through time-driven monitoring) for interesting changes.

Events for individual objects or groups of objects are gathered into traces that describe system behavior over time, and those traces are often stored for later analysis. Management of large volumes of event traces can itself be a difficult problem, and schemes for treating logging services as managed objects have been proposed [19]. Monitoring systems typically provide facilities to combine traces, called *trace merging*, to give the network manager different views of the system. For example, traces generated by individual objects may be combined to give a more global view of the behavior of a group of objects. In addition, a single trace can be decomposed into several smaller traces to focus on particular aspects of the trace, such as events with a given priority or from a given source.

For event traces and other types of network state information to be of use to humans in making management decisions, that information must be accessible through a user interface. One common interface is a simple textual display with appropriate formatting (e.g. indentation and highlighting) of state information [17]. Increasingly, textual displays are being replaced by graphical displays. In particular, animation is becoming common. The display contains graphical images such as dials, histograms, pie charts and meters that convey a snapshot of the network's state. As state changes occur and the display is updated, rotating dials and sliding meters provide an intuitive sense for the way in which system state evolves over time. Examples of systems that use animation include SMART and VISIMON [22], Radar [20] and the Test and Measurement Processor (TMP) [11]. More sophisticated functions that are finding their way into products for network managers include browsing through information at different levels of abstraction [11, 18], interactive playback of historical state information [22, 20], and exploration of network state in virtual reality environments [4].

4.2 Event Correlation

Event correlation is the interpretation of multiple events as a unit. Although event-driven monitoring is commonly preferred to time-driven monitoring because of the inherent reduction in information, large networks may still generate thousands of events each day. Event correlation techniques attempt to identify patterns or trends in network alarms so that multiple manifestations of a single fault can be treated as a unit. Such groupings of events can be used as patterns of activity that are constantly being matched against the current state of the network. If any patterns matches, then the fault associated with that pattern is identified as being present. Event correlation is particularly useful for faults that do not directly trigger events, but are only indirectly identifiable by the problems they cause in other portions of the network. In addition to its direct application to fault identification in the form of a rule base, event correlation is also commonly used to reduce the amount of information presented to network managers. If sets of events commonly co-occur (perhaps by way of being causally related), then correlating those events and reporting a "meta-event" significantly reduces the level of detail of the information presented without significantly impacting its content.

Many network management systems in use today include event correlation components. IMPACT is a management platform devoted to correlational tasks as part of the GTE Telecommunications Services NetAlert system [15]. IMPACT includes a graphical user interface that allows domain experts to define *correlation rules*. The left-hand-sides of correlation rules are boolean functions of network state (events). When a left-hand-side matches the current network state (evaluates to true) the action in the rule's right-hand-side is taken. Actions can either assert or clear correlations. Correlation assertions are meta-events that can in turn be matched in the left-hand-sides of other rules. The event correlation system describe in [16] automates certain aspects of the problem by exploring hypotheses about the relationships between events to form correlations and to infer causality. In particular, events are hypothesized to be related if the objects generating the events are adjacent in the protocol stack, if they are at endpoints of a connection, or if they are part of separate connections that share some intermediate resources (e.g. a router or gateway). These hypotheses drive the collection of more detailed information from relevant portions of the network. Once events are determined to be related, heuristic reasoning about causality relationships is performed based on the type, severity, origin and level in the protocol stack of the events. Finally, more sophisticated and theoretically sound approaches to reasoning about the presence of faults based on multiple sources of uncertain evidence (events from multiple points in the network) are beginning to be explored. In [5], the authors use a Bayesian inference model in combination with a probabilistic belief updating algorithm to perform fault identification and diagnosis in Linear Lightwave Networks.

4.3 Automating Fault Identification

The technology most commonly used to add significant levels of automation to network management platforms is the rule-based expert system. Expert systems employ generic reasoning engines to operate on rule bases that embody domain specific expert knowledge. They have enjoyed considerable success in at least three areas of network management: maintenance, provisioning and administration [3]. Maintenance and administration include facets of fault management such as monitoring, troubleshooting, diagnosis, traffic management and fault remediation. The most common application of expert systems is in the realm of monitoring and diagnosis, with more than a dozen deployed systems. The Automated Cable Expertise (ACE) system is used by several of the regional Bell operating companies to identify faults in local loop plants. GTE's Central Office Maintenance Printout Analysis and Suggestion System (COMPASS) is used to identify faults in the telephone exchanges they manufacture. AT&T's Network Management Expert System (NEMESYS) provides real-time monitoring and control of network traffic patterns.

Although expert systems are a common component of network management software, the problems of brittleness and the knowledge acquisition bottleneck that were discussed in previous sections still limit their utility. Expert systems perform well within the confines of their knowledge base, but tend to fail horribly in new and different circumstance. In addition, rapid changes in technology and the configurations of individual networks ensure that knowledge bases typically have limited lifetimes, bringing the knowledge acquisition problem to the fore. Increasingly, researchers are turning to techniques from the field of machine learning to develop adaptive systems that acquire the knowledge they need automatically.

Two topics within machine learning appear to be garnering the most interest from the network management community: neural networks and top-down induction of decision tree. As outlined in Section 4.2, many approaches to proactive network management and information reduction involve correlating network events. The goal is to find patterns of activity that are indicative or predictive of network faults. Neural networks are simply function approximators that can be trained to identify such patterns in their input. Applications reported in the network management literature include prediction of trunk occupancy in telephone networks [8], computation of a single scalar value based on network state that indicates the overall "health" of the network [7], and detection and prediction of chronic transmission faults in AT&T's digital communications network [31]. Decision trees are a different type of function approximator that can learn mappings from network states to categorical values (typically thought of as class labels). For example, one could have a human expert label network state vectors according to the presence or absence of particular types of faults, and use a decision tree algorithm to learn the mapping used by the expert. Systems reported in the literature include Function-Based Induction (FBI) [10], which learns decision trees from examples as described above and converts those trees into rule sets, and ITRULE [9], an information theoretic learning algorithm that was used to acquire expert knowledge from trouble ticket and alarm databases for inclusion into a rule-based expert system.

Machine learning approaches to acquiring knowledge for network management have met with mixed success. All machine learning techniques rely to a large extent on the quality of their inputs. Therefore, when training examples are presented that include all of the features used by human experts in making their judgements, performance tends to be good. However, when complete and informative feature sets are difficult to obtain a priori (perhaps because human experts are incapable of recognizing all of the information they use in making judgements), performance may be poor. For example, in [8] the authors used past values of trunk occupancy to predict future values, with little success.

5 A New Approach to Fault Identification

In this section we propose a new approach to fault identification based on the idea of *network steering*. In previous work with transportation networks we found that network operation can achieve higher levels of stability when pathologies are predicted and avoided rather than detected and repaired [24]. Much as drivers on busy roads monitor for potentially dangerous situations and steer their cars to avoid them, network managers should look for indications of problems that may appear on the horizon and "steer" the state of the network away from such eventualities. Network steering distributes the network manager's work over time, so more resources are available to attend to unpredictable faults when they arise.

5.1 Proactive Fault Identification

Network management can be either a reactive process, set in motion by one or more indicators of an existing problem, or a predictive process, initiated by indicators of the potential for problems in the near future. Consider trap messages generated by agents in response to state changes in managed objects. Traps may be the result of completely unpredictable events, such as the loss of an underground line to a backhoe, a hardware failure in a router, or a software error in a user's application. When a trap message arrives at the manager's console, it is indicative of an existing problem and the only choice is to react; the fault is isolated, the problem diagnosed, and a corrective course of action is formulated.

However, some types of faults within a network are predictable with varying degrees of accuracy. Consider a trap message generated in response to some feature of a managed object's state exceeding a threshold, such as the number of packets dropped by a router due to a lack of buffer space. It may be that the values of that feature as they change over time are correlated with other features of the same object's state or with features of the state of other objects in the network. If one could find such correlations and use them to predict future states of managed objects, then it would be possible to intervene before the threshold is exceeded, and avoid the pathological state that would generate a trap. Note that predicted and existing faults are handled in much the same way. The isolation, diagnosis, and remediation phases following prediction or detection of a fault are the same, and the same mechanisms can be used in both cases. The advantage afforded by a predictive component is that problems are solved before they reach significant levels, thereby keeping the operation and performance of the network more stable.

Our approach to proactive fault identification will employ a statistical learning algorithm, called Multi-Stream Dependency Detection (MSDD), that finds dependencies between values in multiple streams of data over time. Dependencies are unusual co-occurrences of values. MSDD accepts as input a history of vectors of categorical values that describe the state of the network as it changes over time. State vectors can be obtained by polling for data contained in MIBs. From that history, MSDD produces a set of probabilistic rules that describe how various features of network state are predictive of future states of the network. In particular, MSDD learns rules that relate the state at time t to the state at time $t + k$, where k is called the *lag* of the rule and is an input parameter to the algorithm. For the purpose of fault identification, if the state vector contains information about faults that have occurred in the past, MSDD can learn rules that probabilistically relate current network state to future faults. Such rules facilitate proactive network management, alerting network managers to the possibility of trouble in the future. Our approach is faithful to the Internet model of network management in that it places no additional burden on network elements. The MSDD algorithm runs solely at the network management stations, finding dependencies between features of managed objects within the NMS's purview.

At all levels, network management is a knowledge-intensive task. Identifying indicators of pathological states, diagnosing the causes of faults, and formulating corrective plans all require detailed knowledge of both the domain and the problematic network. If that knowledge must be obtained from human experts, knowledge acquisition quickly becomes a bottleneck. The dynamic nature of computer networks only exacerbates the problem. Network topology, usage patterns, and individual software/hardware components change over time, often rendering knowledge gained in one environment or situation incorrect or irrelevant when applied elsewhere. By automating the process of gathering knowledge about how the state of a network changes over time, MSDD makes it possible to easily acquire knowledge tailored to specific sites, and even to update (re-acquire) such specific knowledge as the dynamics of a site change over longer time scales.

To see how the MSDD algorithm might be used to find dependencies in a computer network, consider a simple LAN environment with three hosts supporting user applications and a single router, all of them running the Internet protocols. Agents running within this network will maintain MIBs for each of the managed objects — the individual hosts, the router, specific applications, etc. The MIBs for the hosts, which are all running TCP/IP, will contain variables such as *tcpAttemptFails* and *tcpCurrEstab*, the number of failed connection attempts and the number of currently established connections respectively. The network management station for the network will periodically poll the individual agents for the contents of their local MIBs, building up time series (streams) for the individual MIB variables. For example, the value of the *tcpAttemptFails* variable retrieved from host A at regular intervals will form one stream, as will the values of that variable obtained from the other hosts on the LAN. Likewise, three different streams for the *tcpCurrEstab* values will be obtained over time, one for each of the hosts. In addition to polled values, notifications of events, such as failures or traps generated due to threshold violations, will be placed into yet another stream, the *event* stream. The contents of these streams give a very detailed picture of how the hosts and other network components (e.g. the router) have changed over time. It is these streams that serve as input to the MSDD algorithm. MSDD's job is to find structure in the streams, to find statistically significant dependencies between states of the network over time.

The MSDD algorithm will run periodically (e.g. every night) at the network management station, generating rules that capture structure in the most recent set of streams. Those rules that are discovered off-line will later be used for on-line control of the network. For example, suppose host A often has many bursty applications running concurrently that access a remote server, and that host B usually supports a few applications that require consistently high bandwidth. Then MSDD might find the following rule:

```
(host-A-tcpCurrEstab high), (host-B-tcpOutSegs high) ->
(router-ipInDiscards threshold)
```

That is, large numbers of open connections on host A coupled with high bandwidth traffic from host B often leads to unacceptably high rates of packet loss in the router (e.g. due to insufficient buffer space). Values in the **host-A-tcpCurrEstab** time series would be obtained by polling the agent responsible for host A, whereas information about the **router-ipInDiscards** stream would come from trap messages generated when the established threshold is exceeded. Suppose the preceding rule exists in the current rule base for our LAN, and that it was generated with a lag of 10. As the network management station polls to get current stream values, if it sees **(host-A-tcpCurrEstab high)** and **(host-B-tcpOutSegs high)** then it can conclude that a trap indicating that the **ipInDiscards** threshold at the router has been exceeded will follow within 10 time steps. At that point, the network's normal recovery mechanisms can be used to suggest some course of action to avoid the trap. For example, the network management station could send a *source quench* message to host B, telling it to slow down the rate at which it sends packets onto the LAN.

5.2 The MSDD Algorithm

In this section we describe the MSDD algorithm is detail.

A *dependency* is an unexpectedly frequent or infrequent co-occurrence of events over time. The MSDD algorithm provides a very general and efficient framework for finding dependencies among values in multiple streams of categorical data [26]. MSDD is general in that it performs a simple best-first search over the space of possible dependencies, and can be adapted for specific domains by supplying domain-specific evaluation functions.

MSDD assumes a set of m streams, $\mathcal{S}$, such that the i^{th} stream, s_i, takes values from the set of tokens $\mathcal{T}_\rangle$. A multitoken is an m-tuple that specifies for each stream either a specific value or an assertion that the value is irrelevant. To denote the latter, we use the special wildcard token $*$, and we define the set $\mathcal{T}_\rangle^* = \mathcal{T}_\rangle \cup \{*\}$. A multitoken is any element of the set created by taking the cross product of all of the $\mathcal{T}_\rangle^*$; that is, multitokens are drawn from the set $\mathcal{T}_\infty^* \times \ldots \times \mathcal{T}_\Updownarrow^*$. Consider a two-stream example for which $\mathcal{T}_\infty = \mathcal{T}_\in = \{\texttt{A, B}\}$.

Adding wildcards, $\mathcal{T}_\infty^* = \mathcal{T}_\in^* = \{$A, B, *$\}$. The space of multitokens for this example ($\{$A, B, *$\} \times \{$A, B, *$\}$) is enumerated below:

```
(A A), (A B), (A *)
(B A), (B B), (B *)
(* A), (* B), (* *)
```

The state of the streams, $x(t)$, is given by the values of the tokens in all streams at a given time. Therefore, $x(t)$ is a multitoken drawn from the set $\mathcal{T}_\infty \times \ldots \times \mathcal{T}_\Updownarrow$. We denote a *history* of vectors obtained from the streams at fixed intervals from time t_1 to time t_2 as $\mathcal{H} = \{\S(\sqcup)|\sqcup\infty \leq \sqcup \leq \sqcup\in\}$. MSDD explores the space of dependencies between pairs of multitokens drawn from the set $\mathcal{T}_\infty^* \times \ldots \times \mathcal{T}_\Updownarrow^*$. (Recall that multitokens specify for each stream either a value or the wildcard token.) Dependencies are denoted *prec* $\overset{k}{\Rightarrow}$ *succ*, and are evaluated with respect to $\mathcal{H}$ by counting how frequently an occurrence of the precursor multitoken *prec* is followed k time steps later by an occurrence of the successor multitoken *succ*. k is called the lag of the dependency, and can be any constant positive value. Consider the three-stream history shown below:

```
Stream 1: A D A C A B A B D B A B
Stream 2: C B C D C B C A B D C B
Stream 3: B A D A B D C A C B D A
```

The dependency (A C *) $\overset{1}{\Rightarrow}$ (* * A) is strong. Of the five times that we see the precursor (A in Stream 1 and C in Stream 2) we see the successor (A in Stream 3) four times at a lag of one. Also, we never see the successor unless we see the precursor one time step earlier.

Note that when counting occurrences of a multitoken, the wildcard matches all other tokens; for example, both (D B A) and (C D A) are occurrences of (* * A), but (C D D) is not. A multitoken composed entirely of wildcards matches all other multitokens. There are 12 multitokens in the streams above, and 12 occurrences of (* * *).

MSDD performs a general-to-specific best-first search over the space of possible dependencies. Each node in the search tree contains a precursor multitoken and a successor multitoken. The root of the tree is a precursor/successor pair composed solely of wildcards. For the three-stream example given above, the root of the tree would be (* * *) $\Rightarrow$ (* * *). The children of a node are specializations of that node, generated by instantiating wildcards with tokens. Each node contains all of the non-wildcard tokens that appear in its parent and exactly one fewer wildcard than its parent. Thus, each node at depth d has exactly d non-wildcard tokens distributed over the node's precursor and successor. For example, both (A * *) $\Rightarrow$ (* * *) and (* * *) $\Rightarrow$ (* D *) are children of the root node. In contrast, both (* * B) $\Rightarrow$ (* C *) and (* * *) $\Rightarrow$ (* A A) are children of nodes at depth one.

The space of two-item dependencies is clearly exponential. The number of possible multitokens is given by $|\mathcal{T}_\infty^* \times \ldots \times \mathcal{T}_\Updownarrow^*| = \prod_{\rangle=\infty}^{\Updownarrow} |\mathcal{T}_\rangle^*|$. If each stream contains k distinct tokens (including *) and there are m streams, then the number of possible multitokens is k^m, and the number of possible dependencies is k^{2m}. Given the size of the search space, MSDD requires domain knowledge to guide the search and to allow efficient pruning. Both goals are accomplished by expanding the children of a node systematically to ensure that no dependency is explored more than once. Specifically, the children of a node are generated by instantiating only those streams to the right of the right-most non-wildcarded stream in that node. Figure 1 shows the expansions for nodes that might be generated, given two streams taking their values from the set {A, B}.

```
(* *) -> (* *) :  (A *) -> (* *)  (B *) -> (* *)
                  (* A) -> (* *)  (* B) -> (* *)
                  (* *) -> (A *)  (* *) -> (B *)
                  (* *) -> (* A)  (* *) -> (* B)

(* A) -> (* *) :  (* A) -> (A *)  (* A) -> (B *)
                  (* A) -> (* A)  (* A) -> (* B)

(B *) -> (A *) :  (B *) -> (A A)  (B *) -> (A B)
```

Figure 1: Three illustrative node expansions from a tree in which two streams take their values from the set {A, B}. For each expansion, the parent node is shown to the left of the colon and its children are listed to the right. Note that only wildcarded streams to the right of the rightmost non-wildcarded stream are instantiated.

In addition to ensuring that each dependency is explored at most once, this method facilitates reasoning about when to prune. For example, *all descendants* of the node (* A *) $\Rightarrow$ (B * *) will have wildcards in streams one and three in the precursor, an A in stream two in the precursor, and a B in stream one in the successor. The reason is that these features are not to the right of the rightmost non-wildcard, and as such cannot be instantiated with new values. If some aspect of the domain makes one or more of these features undesirable, then the tree can be safely pruned at that node.

Formal statements of both the MSDD algorithm and its node expansion routine are given in Algorithms 5.1 and 5.2. The majority of the work performed by MSDD lies in evaluating f for each expanded node. Typically, f will count co-occurrences of the node's precursor and successor, requiring a complete pass over $\mathcal{H}$. Assuming that $\mathcal{H}$ contains l vectors of size m, the computational complexity of MSDD is $O(m * l * maxnodes)$.

5.3 Discussion

Our approach to learning rules for the purpose of fault identification with the MSDD algorithm has several advantages. First, MSDD rules allow proactive network management because they probabilistically relate current network state to faults that may be arbitrarily far into the future. By varying the lag at which rules are generated, network managers can obtain rules that look for near-, medium-, and long-term problems. Second, MSDD automates the acquisition of knowledge for proactive network management by learning rules from state histories of the network that is to be managed. Those rules can be re-learned as the dynamics of the network evolve over time, or different sets of rules can be learned for different operating conditions (e.g. one set for weekdays between 8:30am and 9:30am, and another set for the weekend). When knowledge acquisition is automated and fast, it becomes possible to obtain

Algorithm 5.1 MSDD

```
    MSDD($\mathcal{H}$, {,⇕⊣§\≀⌈⌉/)
 1. $expanded = 0$
 2. $nodes$ = ROOT-NODE()
 3. while NOT-EMPTY($nodes$) and $expanded < maxnodes$ do
      a. remove from $nodes$ the node $n$ that maximizes $f(\mathcal{H}, \backslash)$
      b. EXPAND($n$), adding its children to $nodes$
      c. increment $expanded$ by the number of children generated in ($b$)
```

Algorithm 5.2 EXPAND

```
    EXPAND($n$)
 1. for $i$ from $m$ downto 1 do
      a. if $n.precursor[i] \neq$ '*' then
             return children
      b. for $t \in \mathcal{T}_\rangle$ do
           i. $child$ = COPY-NODE($n$)
          ii. $child.precursor[i] = t$
         iii. push $child$ onto $children$
 2. repeat (1) for the successor of $n$
 3. return $children$
```

and experiment with knowledge customized for a variety of situations. In addition, MSDD rules are in a form that facilitates their inclusion into automated reasoning systems, such as rule-based expert systems. Third, because MSDD can learn rules that relate network states to future states, not just future faults or events, it becomes possible to automatically acquire a model of how network state can be expected to change over time. Such knowledge can be used for purposes other than fault identification. For example, rules learned by MSDD could serve as the initial model of a simulation of the network, which in turn could become a component of a more complete management system [32]. Finally, our approach is faithful to the Internet model in that we place all of the computational burden at the network management stations, and the state vectors required to drive the algorithm can be obtained via periodic polling (or through single requests from a MIB such as RMON [33]).

Our approach is similar to some of the event correlation techniques described in Section 4.2. MSDD attempts to find portions of the state of the network that are correlated in the sense that they are predictive of or may be predicted by other portions of network state. Our approach differs from most current approaches to event correlation in that we consider state information obtained via time-driven monitoring rather than just event driven monitoring. (The work described in [16] considers more detailed state information, but only after particular events are hypothesized to be correlated.) The former information allows reasoning about the nature of existing problems as manifested by various events; the latter information allows reasoning about the nature of problems that are likely to appear in the future and their expected manifestations. Our approach is perhaps most similar to work that applies inductive techniques (e.g. decision trees) to learn various types of rules from network data. Inductive methods can be applied just as we apply MSDD to learn predictive rules. However, the rules learned by MSDD are fundamentally more expressive than those learned by all of the inductive learning algorithms reported in the literature. Specifically, MSDD was designed to learn rules with conjunctive *right-hand-sides* as well as conjunctive left-hand-sides. Therefore, MSDD can be used to learn much more complex and informative models from network data.

There are several open issues that need to be resolved before we will know if our approach will be useful in the large, complex networks that are common today. Most of those issues relate to the question of scale. While we gain expressiveness and power by considering more than just events in network state, we open the door to the flood of information that event-driven monitoring was designed to ward off. Previous work with the MSDD algorithm has shown that it does an excellent job of finding relevant features for specific problems [23, 25], but it is unclear whether the algorithm will be computationally feasible with extremely large state descriptors.

6 Conclusions

In this paper we have explored fault identification, one aspect of the general network management problem, in great detail. We began by defining the fault identification task, examining ways in which approaches to that task commonly vary, and looking at characteristics of the problem domain and the problem itself that make fault identification difficult. We reviewed both Internet and OSI standards that are relevant to fault identification in that they specify the structure, content and methods for manipulation of network state information in a vendor-independent manner. Although those standards are an important step toward integrated network management, they leave unanswered questions related to how network state information should be used for the purposes of making network management decisions. Therefore, a substantial portion of the paper was devoted to looking at current approaches to fault identification, including work in network monitoring, event correlation, and automation of knowledge acquisition and reasoning for the purpose of fault identification. Finally, we presented a new approach to fault identification and proactive network management based on the MSDD algorithm for learning dependencies among values in multiple streams of data. Advantages of our proposed approach include the ability to predict faults rather than simply react to them once they occur, automated acquisition of the relevant knowledge, automated acquisition of a model of the way network states evolve over time, and faithfulness to the Internet model of network management (currently the most widely accepted model).

The factors that make network management a difficult problem today will only make it a more difficult problem for the foreseeable future. The complexity of modern networks will increase along with the number of users, services, quality-of-service requirements and devices that are supported. In addition, as network bandwidth increases and geographically distant locations come to rely increasingly on network communications, the $bandwidth * delay$ product grows as well. The result is that management decisions become increasingly time critical due to the volume of data that traverses network links and the speed with which network state information becomes out of date. Although integrated network management is beginning to enjoy success in the marketplace, the challenges that lie ahead will ensure that this is an active area of research and development for quite some time.

References

[1] Cynthia Bournellis. Internet '95. *Internet World*, 6(11):47–52, 1995.

[2] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol. Request for Comments 1157, May 1990.

[3] Robert N. Cronk, Paul H. Callan, and Lawrence Bernstein. Rule based expert systems for network management and operations: An introduction. *IEEE Network*, 2(5):7–23, 1993.

[4] Laurence Crutcher and Aurel A. Lazar. Management and control for giant gigabit networks. *IEEE Network*, 7(6):62–71, 1993.

[5] Robert H. Deng, Aurel A. Lazar, and Weiguo Wang. A probabilistic approach to fault diagnosis in linear lightwave networks. In H. G. Hegering and Y. Yemini, editors, *Integrated Network Management, III*, pages 697–708. Elsevier Science Publishers B.V., 1993.

[6] L. Feldkuhn and J. Erickson. Event management as a common functional area of open systems management. In *Proceedings of the First IFIP Symposium on Integrated Network Management*, pages 365–376, 1989.

[7] German Goldszmidt and Yechiam Yemini. Evaluating management decisions via delegation. In H. G. Hegering and Y. Yemini, editors, *Integrated Network Management, III*, pages 247–257. Elsevier Science Publishers B.V., 1993.

[8] Rodney M. Goodman, Barry Ambrose, Hayes Latin, and Sandee Finnell. A hybrid expert system/neural network traffic advice system. In H. G. Hegering and Y. Yemini, editors, *Integrated Network Management, III*, pages 607–616. Elsevier Science Publishers B.V., 1993.

[9] Rodney M. Goodman and H. Latin. Automated knowledge acquisition from network management databases. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management, II*, pages 541–549. Elsevier Science Publishers B.V., 1991.

[10] Shri K. Goyal. Knowledge technologies for evolving networks. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management, II*, pages 439–461. Elsevier Science Publishers B.V., 1991.

[11] D. Haban and D. Wybranietz. A hybrid monitor for behavior and performance analysis of distributed systems. *IEEE Transactions on Software Engineering*, 16(2), 1983.

[12] Jayant Haritsa, Michael Ball, Nicholas Roussopoulos, John Baras, and Anindya Datta. Design of the MANDATE MIB. In H. G. Hegering and Y. Yemini, editors, *Integrated Network Management, III*, pages 85–96. Elsevier Science Publishers B.V., 1993.

[13] International Standards Organization - ISO. Information technology - open systems interconnection - common management information protocol - part 1: Specification. volume ISO/IEC 9596 - 1, November 1990.

[14] International Standards Organization - ISO. Information technology - open systems interconnection - common management information service definition. volume ISO/IEC 9595, November 1990.

[15] Gabriel Jakboson and Mark D. Weissman. Alarm correlation. *IEEE Network*, 7(6):62–71, 1993.

[16] J. F. Jordaan and M. E. Paterok. Event correlation in heterogeneous networks using the OSI management framework. In H. G. Hegering and Y. Yemini, editors, *Integrated Network Management, III*, pages 683–695. Elsevier Science Publishers B.V., 1993.

[17] J. Joyce. Monitoring distributed systems. *ACM Transactions on Computer Systems*, 5(2):121–150, 1987.

[18] J. Magee Kramer and K. Ng. Graphical configuration programming. *IEEE Computing*, pages 53–65, 1989.

[19] Lee LaBarre. Management by exception: OSI event generation, reporting and logging. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management, II*, pages 227–242. Elsevier Science Publishers B.V., 1993.

[20] R. J. LeBlanc and A. D. Robbins. Event driven monitoring of distributed programs. In *Proceedings of the Fifth International Conference on Distributed Computing Systems*, pages 512–522, 1985.

[21] Keith McCloghrie and Marshall T. Rose. Management information base network management of TCP/IP based internets. Request for Comments 1156, May 1990.

[22] B. Mohr. SIMPLE: A performance evaluation tool for parallel and distributed systems. In *Proceedings of the Second European Distributed Memory Computing Conference*, pages 80–89, 1991.

[23] Tim Oates. MSDD as a tool for classification. EKSL Memorandum 94-29. Department of Computer Science, University of Massachusetts at Amherst, 1994.

[24] Tim Oates and Paul R. Cohen. Toward a plan steering agent: Experiments with schedule maintenance. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 134–139. AAAI Press, 1994.

[25] Tim Oates and Paul R. Cohen. Learning planning operators with conditional and probabilistic effects. In *Working Notes of the AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, pages 86–94, 1996.

[26] Tim Oates, Matthew D. Schmill, Dawn E. Gregory, and Paul R. Cohen. Detecting complex dependencies in categorical data. In Doug Fisher and Hans Lenz, editors, *Finding Structure in Data: Artificial Intelligence and Statistics V*, pages 185 – 195. Springer Verlag, 1995. Includes work on an incremental algorithm not contained in workshop version.

[27] Information processing systems - Open Systems Interconnection. Specification of abstract syntax notation one (ASN.1). International Organization for Standardization, International Standard 8824, December 1987.

[28] Information processing systems - Open Systems Interconnection. Specification of basic encoding rules for abstract notation one (ASN.1). International Organization for Standardization, International Standard 8825, December 1987.

[29] Marshall T. Rose. Management information base network management of TCP/IP based internets: MIB-II. Request for Comments 1213, March 1991.

[30] Marshall T. Rose and Keith McCloghrie. Structure and identification of management information for TCP/IP based internets. Request for Comments 1155, May 1990.

[31] R. Sasisekharan, V. Seshadri, and S. M. Weiss. Proactive network maintenance using machine learning. In *Proceedings of the 1994 Workshop on Knowledge Discovery in Databases*, pages 453–462, 1994.

[32] Padhraic Smyth, Joseph Statman, Gordon Oliver, and Rodney Goodman. Combining knowledge-based techniques and simulation with applications to communications network management. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management, II*, pages 505–515. Elsevier Science Publishers B.V., 1991.

[33] Steve Waldbusser. Remote network monitoring management information base. Request for Comments 1271, November 1991.