Induction and Deduction for Autonomous Networks

Ying-Dar Lin and Mario Gerla

Abstract— **The key issues in network management are the representation and sharing of management information and the automatic management mechanisms based on the underlying information infrastructure. In this paper, we propose a framework, which operates on the standard MIB's and CMIP, for the network management system with learning and inference as its management engines. In addition to the general domain knowledge, patterns related to the managed network are learned to enhance the understanding of the network and refine the knowledge base. Facts in object-oriented databases or queries from management applications trigger the inference process on logical rules which are either prespecified knowledge or learned network patterns. Forward inference drives prediction and control, while backward inference directs diagnosis and supports view abstraction. A case study on ATM network topology tuning is presented.**

I. Introduction

It is recognized that patterns exist in the behavior and interaction of entities in a complex system. A network is a complex system with various types of traffic sources where traffic occurs due to the interactions between these traffic sources. Network patterns are influenced by the way traffic sources behave locally and interact pairwisely with each other. In general, a network pattern can exist in client–server interactions, temporal and geographical traffic distribution, traffic/performance relationship, performance correlation between network entities, alarms or faults correlations, and some hidden causal relationships. In another study [1], the authors propose a new model for packet arrivals in a short time scale. Here, we are looking at a longer time scale and assume that we do not have a model for the behavior of traffic sources.

The importance of understanding and, furthermore, capturing patterns stems from several different reasons. Phenomena can be explained more precisely, and problems can be diagnosed. Knowing the dynamics within the system will enable us to predict the system behavior and perform adaptive control. In an adjustable system, we can further tune the system according to the pattern if some status is foreseen to occur.

To understand the network patterns, we need the historical information of the network. Three issues arise here. First is the representation issue: in what format are we going to store the current/historical information and the discovered patterns? Second is the learning or knowledge acquisition issue: how are we going to discover the patterns from the stored information trace? Third is the inference or knowledge use issue: based on the management information and the captured patterns, what kind of automatic control/management mechanisms can be built?

Recent progress in network management is the recognition of the need to use standardized databases for storing network management information and a standardized protocol to access the stored information. This solves the interoperability problem [2]–[6]. Definition and implementation of MIB (Management Information Base) and CMIP (Common Management Information Protocol) are ongoing efforts [7], [8]. However, little was done to define the management application schemes based on this common platform.

In this paper, we propose a framework for a network management system with learning and inference abilities, where learning is to capture network patterns and inference is to reason on the discovered patterns and prespecified knowledge in order to access virtual global objects, predict network status, trigger control actions, and diagnose problems. The proposed scheme is meant to operate on the standard management architecture where management information is stored in object-oriented databases. Management knowledge base which includes network patterns, abstract view definition, and domain knowledge is represented as a set of logical rules. These rules are triggered by the facts in databases and queries from management applications. The goal is autonomous network management by expert systems with learning capability.

Section II highlights the network management issues and their recent progress. The induction/deduction approach is proposed in Section III. In Section IV, network patterns are classified and the pattern discovery process is described. The backward deduction for diagnosis and abstraction, and the forward deduction for prediction and control are illustrated. The architectural aspects of the proposed scheme and its operation on the standard management model are described in Section V. The techniques to build management information infrastructure and management applications are detailed. Section VI presents an example on ATM network performance management.

II. Network Management Problems

Unlike real-time control, management is not an essential component to simply make the system work. That is, a system can continue to function, at least for a period of time, without the management subsystem. However, what were once highly tuned systems may gradually degenerate to an inefficient state. Not only a software/hardware failure but also performance degradation can be a system problem. Thus, the task of the management subsystem is to keep track of the system status, which includes both configuration and performance,

Manuscript received May 29, 1992; revised March 7, 1993. This research was supported in part by grants from Mitsubishi and NSF-INT 9115882.

Y.-Q. Lin is with the Department of Computer and Information Science, National Chiao-Tung University, Taiwan.

M. Gerla is with the Department of Computer Science, University of California, Los Angeles, CA 90024.

IEEE Log Number 9212155.

and trigger control actions when necessary. We can divide the management process into the monitoring process and the control process. The monitoring process involves collecting information about the system's short-term/long-term behavior and low-level/high-level status, filtering out unimportant information to reduce stored data volume, and interpreting the semantics of the collected information. The control process affects the state of the system according to the interpreted information to achieve a desired outcome. In this processes, we find that there are two major issues in network management: management information infrastructure and automatic/adaptive management schemes.

A. Management Information Infrastructure

Any network management system must be constructed on top of the underlying management information model on which the representation schemes and operations are based. Given that a network is a distributed, and maybe heterogeneous, environment, several issues are confronted when designing the infrastructure of the network management information.

- *Management information representation:* In what form can the information be stored in network entities and exchanged between network entities? What kind of management information needs to be supported? Do the format and the content have to be standardized for information sharing?
- *Heterogeneity of protocol stacks:* How can machines with different protocols interoperate to share management information?
- *Information distribution strategy:* What is the mechanism for information sharing between network entities and the management system? Should the management system keep a global view of the network at all time or reconstruct it, when needed, from local views of network entities?

Here we are facing problems similar to information sharing problems in a traditional file system, with multiple applications, where an application must know the structure of the files it is operating with. If one particular application needs to modify the structure of a file, all the other applications using that file have to be changed. The solution to avoid this led to the evolution of database systems which contain the files, the file structures, and the primitives to access them. The separation between data and applications provides *data independence* for applications [9]. By the same philosophy, data independence for network protocol stacks and management applications can be supported by the management information databases and their access protocol. The database primitives and the access protocol form the information access primitives for protocol stacks and management applications. As information may be shared by distributed heterogeneous network entities, management databases and access protocol must be standardized.

The open networking community has settled on a management model that places a MIB on each network node and manages these MIB's remotely with application-level protocols [5], [6], [10]. The widely accepted OSI management model is illustrated in Fig. 1. An MIB, an abstract image of

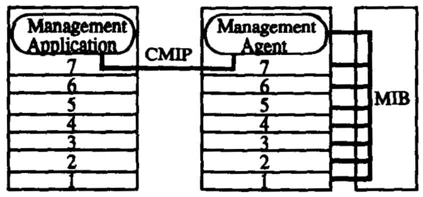


Fig. 1. OSI management model.

the local management objects, is supported by each seven-layer OSI node. Objects are manipulated by the application-layer management protocol CMIP, which uses RPC (Remote Procedure Call) protocol. Changing the attribute values in a MIB will result in changing the status of the physical network entities. For example, setting the status attribute of link 537 to off can disable that link.

Because of the hierarchical nature of network entities and their subentities, both ISO and Internet models organize network management information into a hierarchical structure. ISO even encapsulates this hierarchical model into object-oriented databases in order to hide the heterogeneity of network entities away from the protocol stacks and management applications. In object-oriented databases, the following concepts are supported: i) subtype hierarchy (by record formation and set formation) and method inheritance; ii) encapsulation; and iii) object identity [9]. An *object class* is associated with a set of *methods* operating on the *object instances* of this object class. An object subclass inherits the set of methods from its parent object class. The encapsulation of the heterogeneity of network entities is achieved by the sets of methods.

The adopted architecture solves the problems of *information representation* and *heterogeneity of protocol stacks* , but the problem of *information distribution strategy* remains. Given the standard platform, we still need a mechanism to construct the global views for the management applications. This is one of the problems we want to solve in this paper.

B. Automatic and Adaptive Management

Although the infrastructure of network management is agreed upon regarding the standard MIB's and CMIP, little was done to define how to use this platform in specific network management problems: performance, configuration, fault, accounting, security, etc. Several researchers have adopted expert systems with domain knowledge represented as a set of logical rules capturing network management model to cope with fault localization and correction [11]–[13]. In these systems, network messages containing "trouble tickets" are sent to the expert system. This expert system then reasons on the trouble tickets and network configuration to find the possible fault locations and the recovery procedures. The effectiveness of these systems depends heavily on encoding the problem-solving knowledge in the network domain. The goal of these expert systems is an automatic fault management system to enhance or even replace human intervention.

Other network management problems also need automation. The maintenance of a large number of objects in MIB's needs to be done automatically to keep the status information

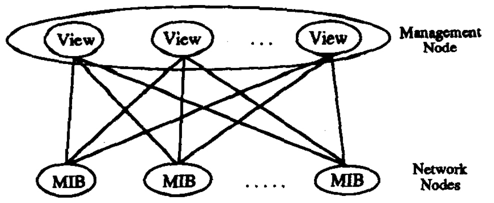


Fig. 2. Information infrastructure: Global views and local MIB's.

up-to-date. Configuration management applications can then easily identify and update objects. This, in turn, changes the configuration of network entities. Either remedial or preventive performance management schemes need to be triggered automatically by performance alarms or traffic forecasting, which again depend on automatic interpretation of performance and traffic measurements. This measurement interpretation implies that the system needs to keep track of the *network patterns* and perform *adaptive control*. The ultimate goal for network management should be a self-managed and self-adjustable network.

III. Approach

Our approach to solve these two network management problems: information distribution strategy and automatic/adaptive management is to incorporate learning and inference abilities into network management systems to automate the process of global view construction, measurement interpretation, problem forecasting, problem diagnosis, and decision making. To build the information infrastructure, a set of global views is constructed. A *global view* is a virtual object class defined from all local MIB's via logical rules. These global views serve as *windows* through which management applications can access physical network entities. Fig. 2 shows a set of global views constructed from local MIB's. To equip the system with automatic and adaptive abilities, network patterns are learned from a historical database which contains a chronological measurement trace. These discovered patterns, represented in the form of logical rules, describe the correlation between network objects. Based on these network patterns and prespecified domain knowledge, forward and backward inference can be triggered to access global views, predict network status, fire control actions, and diagnose reported problems. Fig. 3 illustrates the general approach using learning and inference in network management. Unlike an expert system with only prespecified domain knowledge, the proposed management system has, in addition, learning ability to augment its knowledge regarding the specific managed network.

Fig. 4 is an abstract information flow model of our management systems. EDB's (Extensional Databases) are actually the standard object-oriented MIB's. They represent the basic facts about configuration, traffic/performance measurements, and events/alarms of local nodes. Each network node has an associated EDB which is its local view about the network. IDB (Intensional Database), located at a management site, is defined as the deductive closure of EDB's with logical rules.

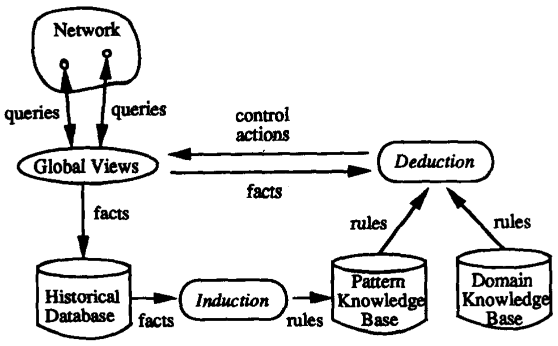


Fig. 3. Induction/deduction in network management.

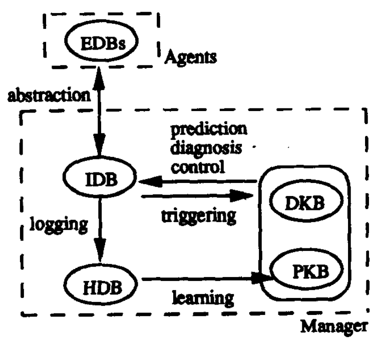


Fig. 4. Abstract model of information flow.

That is, IDB contains virtual objects defined on the physical objects in EDB's. Access to IDB will be transformed into access to EDB's. This is the same concept as in relational databases, where views are virtual relations defined on physical relation tables. EDB and IDB are both deductive database terminologies [9]. The difference is that now IDB is defined on *distributed* EDB's. IDB, including overall configuration and interobject relationships, embodies the global views of the network. Extracted from IDB, HDB (Historical Database) is the temporal historical database which encodes time in the network trace. Network patterns are learned from HDB and stored in PKB (Pattern Knowledge Base). DKB (Domain Knowledge Base) is prespecified problem solving and general relationship knowledge. Note that only EDB's are standardized; all the others are management application dependent.

A logical rule in IDB/PKB/DKB has the generic form: IF X THEN Y, where X is its *body* part and Y is its *head* part. A body or head part has one, or more than one, *formula* which can represent the status of a network object or an action to update an object's status. A detailed definition of logical rules in IDB/PKB/DKB is given in the next section.

Each network pattern, represented as a logical rule in PKB, describes a correlation between the attributes of network objects. These correlations are extracted from HDB, where selected attributes are logged according to the specific management application. Since this extraction is a statistical process, a probability is associated with each logical rule to show how strong this pattern is.

If the status of network objects satisfies the body part in the rule, the pattern tells us, from past experience, that it is very likely that the status of the network object also satisfies the head part with some probability. This logical rule is, thus, fired as a forward inference. Forward inference is very suitable for status prediction. If some undesired status of a network object is foreseen to occur, it can further fire some logical rules in DKB and then trigger preventive control actions. On the other hand, if trouble is reported to the management system (e.g, blocking probability of connection 839 is larger than 5%), this object associated with the trouble is matched against the object in the head of rules. If the head is satisfied, the rule is fired as a backward inference and a series of inferences on the formulas in the body can carry on. Finally, the set of residual formulas which cannot be further deduced are the possible causes to that trouble. Again, using forward inference on the logical rules in DKB, the remedial control actions can be triggered.

Why Rule-Based Systems for Network Management?

After presenting this methodology, let us examine the reasons to adopt rule-based systems for autonomous network management. The following characteristics of network management problems make the rule-based solution desirable:

- Evolving problem-solving knowledge and changing network patterns
- Complex condition matching
- Solution and pattern naturally expressed in IF–THEN rules.

New services or solutions are introduced from time to time, which result in updating problem-solving knowledge. Moreover, different network patterns exist in different networks and they may change over time. Since rules are modular pieces of information that are not explicitly directed by control statements in the program, it is possible to add or remove rules without changing the overall structure of the program or the control flow. In the procedural systems, on the other hand, these changes in problem-solving knowledge may result in modification, recompilation, and reinstallation of the program code. In addition, rule-based systems are powerful in symbolic manipulation by pattern matching between data and rules. A complex situation encoded by many pieces of data can be matched with a set of rules, which is then fired to trigger control actions. It is also more common and natural for network domain experts to express their expertise as declarative IF–THEN rules, rather than as procedural algorithms. Besides, the network patterns which represent cause–effect correlations are naturally expressed as logical rules.

IV. Induction and Deduction in Network Management

A. *Terminologies for Logical Rules*

As mentioned previously, a rule has the generic form: IF X THEN Y. However, the actual formats and meanings of rules in IDB, PKB, and DKB are different. Here, we give the definition of our IDB rule, PKB rule, and DKB rule.

An *IDB rule* is written in the form of Horn clauses, which are statements of the form: "if $A_1, A_2, \cdots$, and A_n are true, then B is true. "Following the Prolog [14] syntax, it is written as:

$$B{:-}A_1, A_2, \cdots A_n$$

where the formulas B and A_i are *predicates* with a list of arguments, e.g. $p(X_1, \cdots, X_k)$. Predicates produce true or false as a result; i.e., they are Boolean-valued functions. A predicate can represent a physical object class stored in EDB's, which is called *EDB predicate*, or a virtual object class defined by IDB rules, which is called *IDB predicate* . IDB rules are used in a backward-chaining fashion to support view abstraction. That is, a query expressed as the predicate B will be transformed into a set of queries/predicates $\{A_i\}$ according to the IDB rule.

The format of a *PKB rule* is in Horn clauses, with certainty factors, like

$$\text{Confidence Factor} = P\% \qquad B \leftarrow A_1, A_2, \cdots A_n$$

which reads "if $A_1, A_2, \cdots$, and A_n are true, B is concluded to be true with probability P." A formula A_i or B is a *condition* that represents the status of a network object (e.g., connection status = "closing," $40\% \leq$ link utilization $\leq 60\%$). PKB rules are triggered in a forward-chaining fashion for status prediction. A transformation from PKB rules to DKB rules is required when PKB rules are to be included in the production system for inference.

A *DKB rule* is actually a production rule. It is written in the OPS5 (Offical Production System, version 5) [15] syntax as:

$$(A_1\, A_2 \cdots A_n \rightarrow B_1\, B_2 \cdots B_m)$$

which reads "if $A_1, A_2, \cdots$, and A_n are true, $B_1, B_2, \cdots$, and B_m will be executed." Here, A_i is a *condition* and B_j is an *action.* DKB rules can be used to invoke control actions by forward inference. They can also emulate backward inference to diagnose problems [16].

B. *Induction for Pattern Discovery*

Learning is a process of knowledge acquisition. Knowledge can be acquired through taking advice, (i.e., inputting new knowledge directly), problem-solving experience (i.e., remembering the structure of the problem and the methods used to solve it), learning from examples (constructing concept definition from examples), etc. [17]. Network measurements are themselves examples containing many implicit, network-dependent patterns to be discovered. The inductive learning constructs decision trees from a large number of examples. Each decision tree represents a concept with the following definition [18].

Definition: A concept i includes the function f_i to be approximated, the set of approximators A_i, the domain D_i ($D_i \subseteq$ HDB), on which f_i and the members of A_{ij} are defined, and the confidence factors, CF_i, which is the percentage of examples in D_i that satisfy the following rule:

$$f_i(D_i) \in [l'_{ij}, u'_{ij}] \leftarrow A_{ij}(D_i) \in [l_{ij}, u_{ij}]$$
$$\text{for all attribute } j$$

where i is the concept index and j is the attribute index. □

As there may be many sets of l', u', l, and u (lower and upper bounds) for a particular set of examples, a set of such rules can be generated from a concept (decision tree). Computational complexities for these learning algorithms are usually exponential in the number of attributes. However, there are steps to reduce complexity by using domain knowledge to restrict the set of attributes and relational structures considered. As learning algorithms are not the main theme of this paper, readers are referred to the literature [17]–[22].

In our approach, induction is performed on the management application dependent HDB to generate PKB. The logical rules in PKB model and represent the correlations between attributes in HDB. Reference [23] reports an experiment on interconnected LAN's where traffic patterns are learned by a machine learning tool from traffic measurements stored in a HDB implemented as a relational database. The discovered rules can describe traffic patterns in terms of locality, long-term burstiness, correlation, cyclic repetition, and predictability. These patterns can be used for medium-term and long-term performance management.

In addition to traffic patterns, there are many other interesting network patterns. In general, patterns describe interobject and intraobject relationships. Here, an object instance is an example. We classify the network patterns into the following categories with examples.

- Temporal and geographical traffic distribution

    Confidence Factor = 85%
    20M $\leq$ Traffic $\leq$ 30M
    $\leftarrow$
    11:30AM $\leq$ Time $\leq$ 12:30PM,
    Source = "oahu",
    Destination = "maui";

- Traffic/performance relationships

    Confidence Factor = 90%
    Delay Violation $\geq$ 5%
    $\leftarrow$
    CPU Utilization $\geq$ 60%,
    Network Application Weight $\geq$ 40%;

- Performance correlation between network entities

    Confidence Factor = 80%
    40% $\leq$ Node B Utilization $\leq$ 50%
    $\leftarrow$
    50% $\leq$ Link 1 Utilization $\leq$ 60%,
    35% $\leq$ Link 2 Utilization $\leq$ 50%;

- Hidden causal relationships

    Confidence Factor = 84%
    Node 3 Fails
    $\leftarrow$
    Number of Performance Alarms from Link 1 $\geq$ 10,
    Link 1 Utilization $\leq$ 20%;

C. Deduction: Forward and Backward Inferencing

Both preventive and remedial control actions can be taken by network management applications. Preventive control is triggered by problem forecasting based on previous patterns, while remedial control is triggered by network events (performance alarms and device failures). As the manager receives results of the queries to IDB, it passes the configuration status variables to configuration submanager, performance status variables to performance submanager, and event variables to fault submanager. If any match between the variable values and the body of a rule occurs, the rule is fired and the head part executed. A rule in IDB/DKB/PKB can be fired for four possible purposes:

- *Prediction:* The forward inference on a PKB rule, given that the rule body is true, forecasts that the rule head will be true.
- *Control:* The forward inference on a DKB rule triggers the control actions to take when some network phenomena are detected.
- *Diagnosis:* The backward inference on a DKB or PKB rule can discover the root causes of network events, even when these events are not yet detected.
- *Abstraction:* The backward inference on an IDB rule transforms an IDB query to EDB query/queries and, hence, provides view abstraction.

Here are two example inference processes: i) a process that predicts traffic demands between node X and Y, forecasts performance alarms for link L, and takes actions to reroute some traffic from link L; ii) a process that diagnoses the received performance alarms, concludes that node Z is malfunctioning, reroutes traffic that passes node Z, and disables node Z.

Backward inference is triggered by events (i.e., only when there are network problems: performance alarms and device failures) and queries (from manager to IDB). However, forward inference is triggered by a set of state variables. The workload on forward inference process can be very high since each state variable will match against each formula in the rule bodies to see if some rules can be fired. Thus, keeping the number of state variables for triggering forward inference small is critical in designing management applications.

V. Architecture

This section describes how the proposed learning and inference schemes work on the standard OSI management platform, the organization of object classes in EDB and rule classes in IDB/PKB/DKB, and the rule-based learning expert system.

A. Distributed Management Architecture

Fig. 5 shows a management system with a manager and several remote agents. An agent resides on each OSI node and manages its MIB (EDB in our terminology). The manager has submanagers (configuration, performance, and fault inference modules in this case) for specific management functions. Periodically, the manager issues a query to IDB, which in turn is forwarded and translated to the EDB's to get management

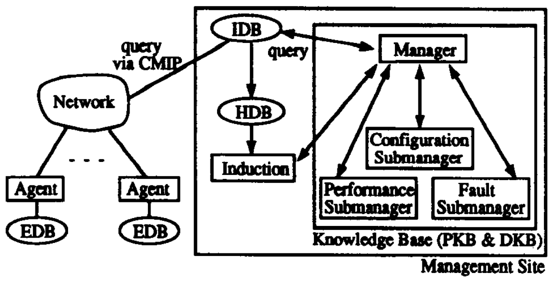


Fig. 5. Manager, submanagers, and agents.

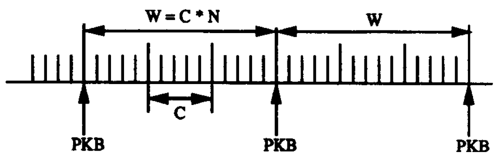


Fig. 6. Induction and query periods.

information, and stores it in the HDB. The results of the query are also passed to submanagers to trigger the inference process on PKB and DKB, if any match occurs. If control actions are to be fired as a result of the inference, the manager updates the corresponding views in IDB, in turn objects in EDB's, through the sets of methods associated with the objects. These updates on EDB's then propagate to the network entities as control actions. Note that a query via CMIP can be a read as collecting management information or a write (create, delete, modify) as taking control actions.

Induction on HDB is also carried out periodically, but much less often, to renew PKB. A sliding window mechanism is used to maintain the consistency between HDB and PKB. That is, HDB only contains records in the most recent W (window size) query periods. An induction is triggered if PKB was generated W query periods ago. Fig. 6 illustrates the relationship between induction period and query period. If C is the number of query periods in a cycle and N is the number of cycles in the window of HDB, $W = C * N$. If there is a temporal repetition in network behavior, cycles exist as network patterns.

Conceptually, management information and knowledge are spread in the layer structure and contained in various databases and knowledge bases, as shown in Table I.

This is similar to the hierarchical blackboard architecture used in signal-processing expert systems [24]. The control strategy, which is implemented as the manager, decides when to execute the rule sets, which are implemented as the submanagers, in configuration, performance, and fault domains. Usually, this is triggered by either status variables or queries. An inference process on DKB and PKB then accesses the objects of a view in IDB, which in turn accesses the remote objects in EDBs over the network. The hierarchy is organized as Fig. 7. The following two subsections describe how to build the views, namely the construction of IDB from the underlying EDB's, and the rule-based management applications based on this infrastructure.

TABLE I
TRAFFIC FOR AN EIGHT-NODE NETWORK

	A	B	C	D	E	F	G	H
A	0	4	1	2	1	4	0	2
B	4	0	48	36	18	0	0	24
C	1	48	0	1	1	373	353	0
D	2	36	1	0	4	0	2	398
E	1	18	1	4	0	3	225	4
F	4	0	373	0	3	0	23	2
G	0	0	353	2	225	23	0	2
H	2	24	0	398	4	2	2	0

(a) Traffic Matrix

Capture Ratio(%)	26	50	73	88	91	93	95
Promotion Ratio(%)	3.5	7	11	14	18	22	25

(b) Traffic Locality

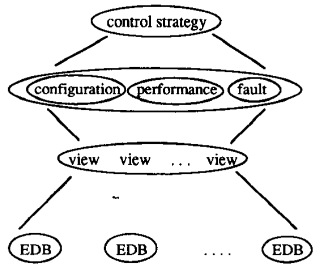


Fig. 7. The blackboard architecture.

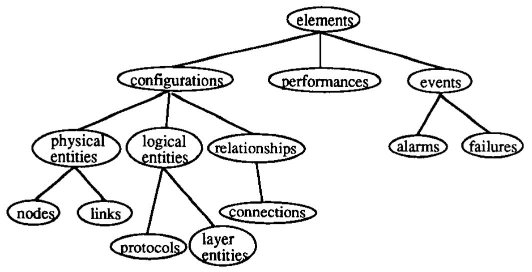
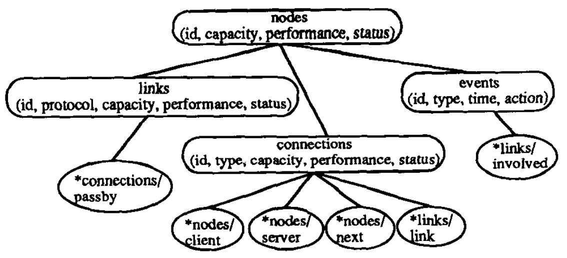


Fig. 8. Inheritance hierarchy.

B. Building Information Infrastructure

Modeling network management information is to map network configuration, performance, and events to objects in EDB's. The *inheritance hierarchy* in Fig. 8 represents a simple classification of network object classes, where the *elements* class has three subclasses: *configurations, performances*, and *events*. *Physical entities* class has two subclasses: *nodes* and *links*.

A node's EDB contains only its *local* management information. Fig. 9 shows an EDB organized in a *containment hierarchy* and its type declaration. An EDB is an object instance of *nodes*. In addition to its own variable attributes, this *nodes* instance contains a set of *links* instances (for links that are connected to this node), a set of *connections* instances (for



```
NodeType = RECORDOF(id: int, capacity:int, performance: PerfType,
status: int, links: SETOF(LinkType), connections:SETOF(ConnectionType),
events: SETOF(EventType));

LinkType = RECORDOF(id: int, protocol:ProtocolType, capacity: int,
performance: PerfType, status: int, passby: SETOF(Connection Type));

ConnectionType = RECORDOF(id: int, type: int, capacity: int,
performance: PerfType, status: int, client: NodeType, server: NodeType,
next: NodeType, link: LinkType);

EventType = RECORDOF(id: int, type: int, time: int, action: ActType,
involved: SETOF(LinkType));

PerfType = RECORDOF(id: int, traffic: int, delay: int, loss: int, interval:int);
```

Fig. 9. EDB: A local MIB.

Manager's Schema for EDBs:

```
l_nodes(Nodeid, Capacity, Perfid, Status, Links, Conns, Events)
l_links(Nodeid, Linkid, Protocol, Capacity, Perfid, Status,Conns)
l_connections(Nodeid, Connid, Type, Capacity, Perfid, Status, Clientid,
   Serverid, Nextid, Linkid)
l_events(Nodeid, Eventid, Type, Time, Action, Links)
l_performance(Nodeid, Perfid, Traffic, Delay, Loss, Interval)
```

Views in IDB:

```
nodes(Nodeid, Capacity, Perfid, Status, Links, Conns, Events)
links(Linkid, Protocol, Capacity, Perfid, Status, Nodes, Conns, Events)
connections(Connid, Type, Capacity, Perfid, Status, Clientid, Serverid,
   Nodes, Links)
events(Eventid, Type, Time, Action, Nodes, Links)
performances(Perfid, Traffic, Delay, Loss, Interval)
```

View Definitions:

```
nodes(Nodeid, Capacity, Perfid, Status, Links, Conns, Events) :-
l_nodes(Nodeid, Capacity, Perfid, Status, Links, Conns, Events).

links(Linkid, Protocol, Capacity, Perfid, Status, Nodes, Conns, Events) :-
l_links(Nodeid, Linkid, Protocol, Capacity, Perfid, Status, Conns),
set_of(N, (member(Linkid, N_links), l_nodes(N, _, _, _, N_links, _, _)), Nodes),
set_of(E, (member(Linkid, E_links), l_events(_, E, _, _, _, E_links)), Events).

connections(Connid, Type, Capacity, Perfid, Status, Clientid, Serverid, Nodes, Links) :-
l_connections(Clientid, Connid, Type, Capacity, Perfid, Status, Clientid, Serverid, _, _),
path(Connid, Clientid, Serverid, Nodes, Links).

path(Connid, End, End, [End], []) :- !.
path(Connid, Start, End, [Start|Noderest], [Linkid|Linkrest]) :-
l_connections(Start, Connid, _, _, _, _, _, _, Nextid, Linkid),
path(Connid, Nextid, End, Noderest, Linkrest).

events(Eventid, Type, Time, Action, Nodes, Links) :-
set_of(Nodeid, l_event(Nodeid, Eventid, Type, Time, Action, _), Nodes),
set_of(Linkid, (member(Linkid, E_links), l_events(Nodeid, Eventid, Type, Time, Action,
   E_links)), Links).

performances(Perfid, Traffic, Delay, Loss, Interval) :-
l_performances(Nodeid, Perfid, Traffic, Delay, Loss, Interval).
```

Fig. 10. Views in IDB.

connections that pass through this node), and a set of *events* instances (for events in which this node is involved). Again, a *links* instance also contains a set of *connections* instances (for connections that pass through this link).

At the management site, what the management applications see is a set of *views*, i.e., a set of IDB predicates. Different sets of views can be defined for different management applications. Each IDB predicate is defined on EDB predicates. The schema at the management site for IDB/EDB predicates and the Prolog implementation to define these IDB predicates are given in Fig. 10. Prolog Logic Programming techniques used here can be found in [14]. Prolog's recursive programming style, which is not supported in relational query languages, provides us a powerful query interface to unify distributed network management information. Prolog emerges as an attractive query language for relational databases [9]. However, with simple extensions, Prolog can also interface with object-oriented databases [25].

To see how an IDB predicate can be constructed by combining EDB predicates, let us take predicate *connections* as an example:

```
connections(Connid, Type, Capacity, Perfid, Status, Clientid,
  Serverid, Nodes, Links)
: – l_connections(Clientid, Connid, Type, Capacity, Perfid,
  Status, Clientid, Serverid, _, _),
path(Connid, Clientid, Serverid, Nodes, Links).

path(Connid, End, End, [End], []) : – !.
path(Connid, Start, End, [Start | Noderest], [Linkid | Linkrest])
: – l_connections(Start, Connid, _, _, _, _, _, _, Nextid, Linkid),
path(Connid, Nextid, End, Noderest, Linkrest).
```

For every "Nodeid," predicate *l_connections* (Nodeid, Connid, Type, Capacity, Perfid, Status, Clientid, Serverid, Nextid, Linkid) contains all connections that pass node "Nodeid." (*l* stands for local.) For every such connection, *l_connections* contains "Nextid" and "Linkid" for its next hop (node and link), but does not know the whole path. *connections* , constructed from *l_connections*, contains the link lists "Nodes" (all nodes on this connection) and "Links" (all links on this connection). "Nodes" and "Links" are constructed by predicate *path* , which takes "Nextid" and "Linkid," starting from the node "Clientid," and inserts them into the link lists "Nodes" and "Links." Note that, in the rule for *connections*, "Nodeid" in *l_connections* is an existential quantifier, which means all nodes can be queried to match with the attributes of *connections*. Similar view construction techniques are used in predicates *links* and *events*. Instead of using recursive predicate *path* , "set_of" constructs are used to construct the link list whose elements satisfy the specified condition. *links* contains "Nodes" (for all nodes connected to this link) and "Events" (for all events involving this link), while *event* contains "Nodes" (for all nodes involved in this event) and "Links" (for all links involved in this event).

All the predicates mentioned are the schema definitions at the management site. An access to a predicate of IDB will be converted, by *backward chaining*, to access to predicate(s) of the manager's EDB's, and then transferred, by CMIP queries, to the physical EDB's on network nodes. Thus, a mapping between access to predicates of the manager's EDB's and CMIP queries to physical EDB's must be done at the management site. The attribute "Nodeid" in each EDB

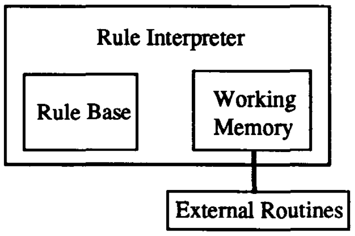


Fig. 11. The rule-based production system.

predicate is used to identify the network node that contains the object instances.

C. Building Management Applications

Based on the constructed IDB, management applications can perform inference on IDB using their knowledge base in DKB and PKB. Data in IDB is matched with the rules in DKB and PKB. Rules can be applied in either direction: forward and backward. The direction corresponds to the type of reasoning and problem-solving strategy. Forward inference is data-driven and bottom-up processing, while backward inference is goal-driven and top-down processing. Prediction and control operations are data-driven, hence forward reasoning. Diagnosis problems are goal-driven, hence backward reasoning.

As shown in Fig. 11, there are four components in the rule-based system: working memory, rule base, rule interpreter, and external routines. The rule-based programming language OPS5 is used to describe how to build the management applications. Although OPS5's inference engine is inherently forward, backward inference can be emulated by treating goals as data and using three sets of rules: a set to split goals into subgoals, a set to recognize and solve achievable subgoals, and a set to fuse the results of subgoals. All these rule-based programming techniques can be found in [15], [16]. The following paragraphs describe the functionalities and operations of these components in our framework.

A. Working Memory: Periodically, a set of rules is triggered to retrieve data from IDB into working memory. A working memory element (WME) is a view in IDB. Event WMEs are treated as goals to trigger diagnosis process, while nonevent WME's are data-to-trigger prediction and control process.

B. Rule Base: In Fig. 7, we have three management applications: configuration, performance, and fault. Each management application is associated with a rule cluster. Rule clusters are in either DKB or PKB. A rule cluster is conceptually equivalent to a procedure. These rule clusters are scheduled by the control rules, which are at the top level of blackboard architecture.

Rule clusters and control rules together perform the periodical management tasks. A period, a management cycle, starts with retrieving data from IDB into working memory and ends when no more data can trigger the rules. In addition to this synchronous management cycle, there are a set of demon rules to perform asynchronous management. Demon rules are not scheduled by the control rules. They can fire any time when an event WME is detected in working memory. An urgent event, like device failure or performance alarm, can be handled immediately by demon rules.

C. Rule Interpreter: Basically, the rule interpreter performs a match-select-act cycle to process WME's. It first matches all WME's with condition elements in all rules, selects one rule with matching WME's, performs actions on the right-hand-side of the chosen rule, and then repeats the cycle.

An event WME is taken as a goal to trigger backward rules to diagnose the root causes of this event. An event WME can be further split into several event WME's until those event WME's are root causes. All other WMe's may trigger forward reasoning if a set of WME's matches the body of a rule.

D. External Routines: The rule-based management applications need to communicate with other management subsystems to access management information and invoke algorithmic routines. To retrieve management information or issue control actions on network entities, queries will be issued to IDB. This is done via external calls to a Prolog program that supports the virtual IDB. Many of the numerical algorithms, like bandwidth allocation and path routing, are not suitable to be implemented in the rule-based language. They are also implemented as external routines.

VI. ATM Network Topology Tuning: An Example

Most of the congestion and flow control procedures for conventional networks cannot be applied to ATM networks, where nodes tend to become the bottlenecks and propagation delay dominates other delays. Congestion (e.g., call blocking, cell loss) is inevitable if there is a mismatch between offered traffic pattern and network topology. This problem can be alleviated by dynamically tuning the topology to traffic pattern. With the technology of digital cross-connect systems (DCS), a broadband packet-switched ATM network can be dynamically reconfigured [26]. The embedded logical topology can be derived from the original physical topology by establishing express pipes between distant nodes. Express pipes and circuit-switched pipes for packet-switched traffic reduce store and forward delay and nodal processing overhead which, in turn, reduces blocking and loss probabilities. In an ATM network without express pipes, all traffic has to be stored and switched at each intermediate node. With express pipes, traffic going through the pipes can "bypass" the intermediate switches. Given the traffic demand matrix, the routing of express pipes and the allocation of bandwidth to such pipes, i.e., embedded topology, can be determined to optimize the GOS (Grade of Service) [27].

In this case study, we will demonstrate how to learn traffic patterns and tune the topology to the discovered patterns, and will show the performance improvement with this scheme. Fig. 12 illustrates the case study. Traffic is generated according to our model, which incorporates the parameters for adjusting locality, burstiness, correlation, cyclic repetition, and predictability. The simulated traffic is fed into a simplified ATM network simulator, and the performance results are evaluated by an analysis module. The management system LEN (Learn-

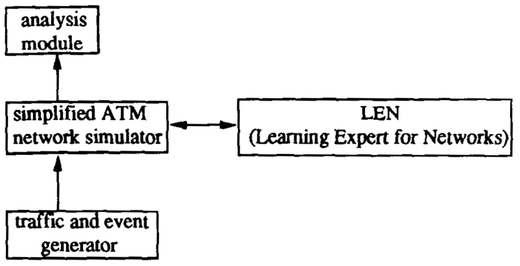

analysis module
simplified ATM network simulator
LEN (Learning Expert for Networks)
traffic and event generator

Fig. 12. Case study on ATM networks.

ing Expert for Networks) performs the management tasks by monitoring the network simulator, learning traffic patterns, and triggering actions to tune the topology. A comparison is made between the performance results for systems with and without LEN. As LEN is still under development, the inference process on the rule base is now done manually.

A. ATM Network Model: Configuration, Traffic, and Operation

1) Configuration: Each node has a switching capacity. Each link between two nodes has a transmission capacity. Similarly, each express pipe between two nodes has a transmission capacity. The schema of nodes, links, and express pipes are the same as those in Fig. 10. Express pipes are represented as the view *connections* in that schema. In LEN, three types of configuration WME's are created: node, link, and pipe. Their attributes are again the same as the ones in Fig. 10.

2) Traffic: To generate a relational traffic table HDB, a base matrix describing bandwidth requirement is generated first, and then N cycles of traffic matrix are generated. A filtering process is imposed on the traffic table to capture only significant traffic components and reduce the stored information volume. i) and ii) are the specifications for traffic generation and filtering.

i) Base matrix generation:

Input: percentages of heavy, medium, and light traffic pairs $H\%, M\%, L\%$

Output: Mean level matrix of bandwidth requirement $MEAN(i,j)$, variance level matrix of bandwidth requirement $VAR(i,j)$

ii) Traffic database generation:

Input: $MEAN(i,j), VAR(i,j)$, discrete usage habit curve $U(t)$, maximum promotion ratio $p\%$, maximum capture ratio $c\%$, number of cycles generated N

Output: Relational Traffic Table HDB (hour, source, dest, bandwidth).

HDB will be the input to our machine learning tool IXL (Induction on eXtremely Large databases) [28]. The learning result is a set of PKB rules.

The defined traffic model reflects the following characteristics: locality, correlation, burstiness, and predictability. Conceptually, $MEAN(i,j), VAR(i,j)$, and $U(t)$ are used to randomly generate a 3D bandwidth requirement matrix $Fij(t)$, where i is source, j is destination, and t is time slot. In the mean time, $p\%$ and $c\%$, defined as $p\%$ of communicating pairs contributing $c\%$ of total traffic, are used as criteria for the promotion process to capture the most significant part of the collected traffic measurements [23]. N cycles of traffic measurements, HDB, will serve as a basis for predicting the traffic distribution of the next cycle.

iii) Induction for traffic patterns:

The inducted PKB rule

```
Confidence Factor = P%
LF ≤ bandwidth ≤ UF
←
START ≤ slot ≤ END,
sourcenode = SRC,
destnode = DST;
```

means the bandwidth requirement of a particular node pair during several continuous slots is between two values with probability $P\%$. The function of IXL is to find out when and how much traffic is flowing from Src to Dest, where time and volume are expressed in terms of ranges. The establishment, at *Start*, and release, at *End*, of pipes are discrete events.

PKB is an abstraction of HDB. It represents the patterns in the past N cycles. According to these inducted patterns, the topology of the ATM network will be tuned with some express pipes established. Each such inducted rule will be automatically transformed into the following two rules and then included into the OPS5 rule base:

```
(p performance!predict-and-create-traffic-WME
      (subtask ^name performance)
      (time ^slot START)
      (node ^name SRC)
      (node ^name DST)
   →
      (make traffic ^predicted-rate (compute-rate PLF UF)
                    ^from START ^until END
                    ^source SRC ^dest DST))

(p performance!delete-traffic-WME
      (subtask ^name performance)
      (time ^slot END)
      { < demand >
      (traffic ^until END)}
   →
      (remove < demand >))
```

3) Operation: LEN is responsible for tuning the topology according to PKB. A management cycle in LEN includes the following steps.

- Retrieve data into WM: call external Prolog program to issue queries to IDB
- Predict traffic demand: create traffic WME's
- Handle traffic WME's: create pipe WME's
- Reconfigure topology: call external Prolog program to write pipe views to IDB.

At the beginning of a time slot, LEN checks if the current slot matches any "START" or "END" entry PKB. If any match occurs, four possible actions can be taken: establish new pipes, augment existing pipes, shrink existing pipes, and release existing pipes. In these four cases, LEN can create new traffic WME's, or modify/delete existing traffic WME's. These traffic WME's will match with a set of rules to create, modify, or delete pipe WME's. Another set of rules then match these pipe WME's to call external routines to physically access the express pipes. The algorithm for pipe bandwidth allocation and pipe routing is described in [27]. The following is an example rule to create a pipe WME from a new traffic WME.

```
(p performance!create-pipe-WME-from-traffic-WME
        (subtask ^name performance)
        {< demand >
        (traffic ^processed? nil
                ^predicted-rate <flow> ^source <src>
                ^dest <dst> ^delay-requirement <delay>)}
        (status ^congestion-level <system-load>)
   →
        (modify <demand> ^processed? YES)
        (make pipe ^bandwidth (compute-bandwidth
                <flow> <delay> <system-load>)
                ^source <src> ^dest <dst>))
```

B. Performance Gain by Learning Traffic Patterns

We now report preliminary results which demonstrate the application and effectiveness of induction and deduction to performance management. While tuning the ATM network in each time slot according to PKB of the previous N cycles, the traffic for the next cycle is generated and applied to the tuned network to compute the connection blocking probability. In the mean time, this probability for the nontuned network is also computed for comparison.

The simulated traffic has a base traffic matrix $MEAN(i,j)$ of Table I(a). The traffic locality is about 14/88 (14% communicating pairs contributing 88% of total traffic) as shown in Table 1(b). The cycle in one day which is divided into 48 time slots. During the period of five cycles, a total of 675 traffic pairs are promoted (with capture ratio set to 90%) and logged into HDB. The induction process takes 2 hours, 29 minutes on a 386 personal computer and generates a PKB containing 74 rules. The example ATM network in Fig. 13 contains 8 nodes, 12 links, and 5 express pipes (three large pipes: C-B-A-D-F, C-E-H-G, E-D-G, and two small ones: B-E-D, B-E-H). In the ATM network simulation, we manually change the topology for each time slot according to PKB and then apply another cycle of traffic. After completion of the LEN implementation, this will be done automatically.

The resulting connection blocking probabilities for nontuned and tuned networks are compared in Fig. 14. The averaged connection blocking probabilities (in percentage) (weighted by traffic volume) under the given load are 6.54% and 1.71% for the ATM network without and with tuning, respectively. The peak blocking probabilities are 11.81% for the nontuned network and 3.57% for the tuned network. The improvement is significant, especially when the network is heavily loaded.

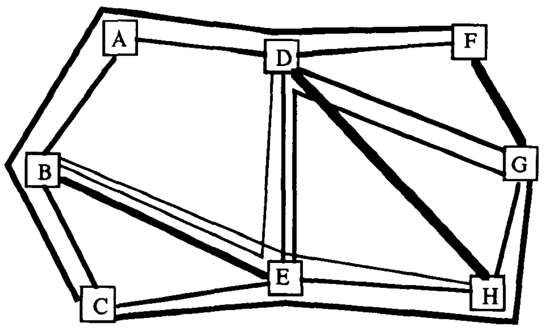


Fig. 13. Connection blocking probability: Non-tuned and tuned.

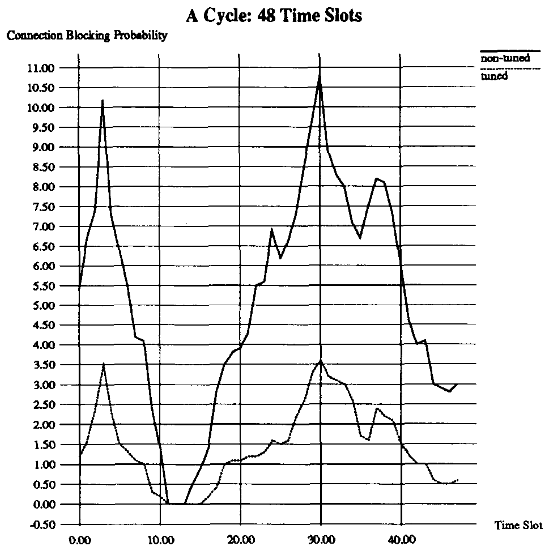


Fig. 14.

VII. Conclusions and Future Work

Inference, as a thinking process on given facts by logical rules, is to find the facts that are not explicitly stated in the knowledge base. That is, the deductive closure K^+ can be derived from the fixed knowledge base K but no more than that. Learning, on the other hand, can expand K^+ by adding facts or rules to K. In our proposed framework, knowledge related to the underlying network is learned to capture network patterns and refine the prespecified domain knowledge. The learning systems have more advanced abilities than nonlearning systems in performance and fault management, which require understanding of traffic patterns and knowledge of causality.

The proposed scheme is designed to operate on the standard platform of MIB's and CMIP. Two main contributions are the global view abstraction, and the integration of learning and inference for autonomous management applications. The case study of ATM logical topology tuning shows significant improvement when dynamic traffic patterns are captured to drive the tuning process. The implementation of LEN (Learning Expert for Networks) is now in progress. Other performance and fault management applications will be built in LEN.

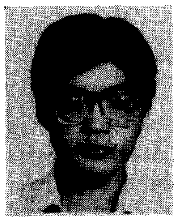
Still, there are several open issues deserving further studies. First of all, the key to success of this approach is to identify the patterns, PKB, to be approximated for the specific management applications. There seems to be no easy way for this, just like domain knowledge of the other expert management systems. Second, fault management systems built on this framework have to rely on DKB if the types of faults have never occurred before. Third, the tradeoff between the extra complexity introduced and the amount of improvement also needs to be evaluated. Finally, this framework has to incorporate probabilistic reasoning techniques [29] since our PKB rules are statistical results.

ACKNOWLEDGMENT

The authors would like to acknowledge their colleagues, S.-T. Huang and C. Zaniolo at U.C.L.A., for the fruitful discussions on issues in deductive databases and logic programming. Thanks to J. Carlyle at U.C.L.A. and J. Monteiro at UFPE in Brazil for their helpful comments during the preparation of the manuscript.

REFERENCES

[1] Y. D. Lin, T. C. Tsai, and M. Gerla, "HAP: A new model for packet arrivals, with implications for broadband network control," im *Proc. ACM SIGCOMM*, San Francisco, CA, Sept. 1993.
[2] J. D. Case, J. R. Davinm, M. S. Fedor, and M. L. Schoffstall, "A simple network management protocol," RFC 1067, SRI Int., Aug. 1988.
[3] K. McCloghrie and M. Rose, "Management information base for network management of TCP/IP-based internets," Internet Stand. RFC 1156, May 1990.
[4] M. Rose, "Management information base for network management of TCP/IP-based internets—MIB II," Internet Stand. RFC 1158, May 1990.
[5] ISO/IEC DIS 10165-1, "Information technology—Open systems interconnection—Structure of management information—Part 1: Management information model," in *Proc. ISO*, Geneva, Switzerland, June 1990.
[6] ISO 9596, "Information technology—Open systems interconnection—Common management information protocol specification," in *Proc. ISO*, Geneva, Switzerland, May 1990.
[7] J. R. Haritsa, M. O. Ball, N. Roussopoulous, and A. Datta, "Design of the MANDATE MIB," in *Proc. Int. Symp. Integrat. Netw. Manage.*, San Francisco, CA, Apr. 1993.
[8] S. Mazumdar, S. Brady, and D. W. Levine, "Design of protocol independent management agent to support SNMP and CMIP queries," in *Proc. Int. Symp. Integrat. Netw. Manage.*, San Francisco, CA, Apr. 1993.
[9] J. D. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, 1988, pp. 11–12, pp. 82–87, pp. 100–101.
[10] L. N. Cassel, C. Partridge, and J. Westcott, "Network management architectures and protocols: Problems and approaches," *IEEE J. Select. Areas Commun.*, vol. 7, no. 7, Sept. 1989.
[11] E. C. Ericson, L. T. Ericson, and D. Minoli, Ed., *Expert Systems Applications in Integrated Network Management.* New York: Artech House, 1989.
[12] S. Goyal, "Knowledge technologies for evolving networks," in Proc. IFIP TC6/WG6.6 Second Int. Symp. Integrat. Netw. Manage., Jan. 1991.
[13] L. Lewis, "A Case-based reasoning approach to the resolution of faults in communications networks," in *Proc. Int. Symp. Integrat. Netw. Manage.*, San Francisco, CA, Apr. 1993.
[14] L. Sterling and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques.* Cambridge, MA: M.I.T. Press, 1986.
[15] T. A. Cooper and N. Wogrin, *Rule-Based Programming with OPS5.* New York: Morgan Kaufmann, 1988.
[16] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming.* New York: Addison-Wesley, 1985.
[17] E. Rich and K. Knight, *Artificial Intelligence, Second Edition.* New York: McGraw-Hill, 1991, pp. 447–484.
[18] J. R. Quinlan, "Induction of decision, trees," *Mach. Learn.*, vol. 1, pp. 81–106, 1986.
[19] P. H. Winston, "Learning structural descriptions from examples," in *Psychology of Computer Vision.* New York: McGraw-Hill, 1975.
[20] J. R. Quinlan, "Generating Production Rules from Decision Trees," in *Proc. 10th Int. Joint Conf. Art. Intell.*, 1987, pp. 304–307.
[21] G. M. Pagallo, "Adaptive decision tree algorithms for learning from examples," Ph.D. thesis, UCSC-CRL-90-27, Univ. of California at Santa Cruz, 1990.
[22] D. Sleeman and P. Edwards, Eds., *Machine Learning: Proceedings of the Ninth International Workshop.* New York: Morgan Kaufmann, 1992.
[23] M. Gerla and Y. D. Lin, "Network management using database discovery tools," in *Proc. IEEE 16th Conf. Local Comput. Netw.*, Minneapolis, MN, Oct. 1991.
[24] B. Hayes-Roth, "A blackboard architecture for control," *Art. Intell.*, vol. 26, pp. 255–321, 1985.
[25] C. Zaniolo, "Prolog: A database query language for all seasons," in *Proc. First Int. Workshop*, 1986.
[26] R. G. Addie and R. W. Warfield, "Bandwidth switching and new network architectures," in *Proc. 12th Int. Teletraff. Cong.*, Torino, Italy, June 1988.
[27] M. Gerla, J. S. Monteiro, and R. Pazos, "Topology design and bandwidth allocation in ATM nets," *IEEE J. Select. Areas Commun.*, vol. 7, no. 8, Oct. 1989.
[28] *IXL: The Machine Learning System, User's Manual.* XXX Intelligence Ware Inc., 1988.
[29] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* New York: Morgan Kaufmann, 1988.



Ying-Dar Lin was born in Taiwan in 1965. He received the Bachelor's degree in computer science and information engineering from National Taiwan University in 1988, and the Master's degree in computer science, from the University of California, Los Angeles, in 1990, where he is completing the Ph.D. degree in computer science.

His thesis work focuses on traffic pattern studies in high-speed networks. At U.C.L.A. Computer Science Department, he works as a Research Assistant and worked as a Teaching Assistant from 1991 to 1992. He is currently Assistant Professor in the Department of Computer and Information Science, National Chiao-Tung University, Taiwan. His current research interests include packet traffic modeling, autonomous network management, connectionless support in ATM networks, and multimedia protocols.

Mario Gerla received the graduate degree in electrical engineering from the Politecnico di Milano, Milano, Italy, in 1966, and the M.S. and Ph.D. degrees in computer science from University of California, Los Angeles, in 1970 and 1973, respectively.

From 1973 to 1976, he was a Network Planning Manager at the Network Analysis Corporation and led several computer network design projects for both government and industry. From 1976 to 1977, he was with Tran Telecommunications, Los Angeles, CA, where he participated in the development of an integrated packet and circuit network. In 1977, he joined the University of California, Los Angeles, and is now a Professor in the Department of Computer Science. His research interests include the design and control of distributed computer communication systems and networks, and the development of protocols for high-speed LAN's, MAN's, and WAN's.