# A Framework for Learning and Inference in Network Management

Ying-Dar Lin and Mario Gerla

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

## Abstract

*This paper presents a network management framework which builds the management information infrastructure and equips the management applications with learning and reasoning abilities for automatic and adaptive management tasks. Views are global virtual management information constructed via logical rules from the distributed physical management information. Through these views, management applications can access physical network entities. Management applications learn network patterns and reason on the discovered patterns and pre-specified domain knowledge to predict network behavior, diagnose problems, and trigger control actions. These abstract view definition, domain knowledge, and network patterns are a set of logical rules stored in the application-dependent MKB (Management Knowledge Base), while the physical management information is stored in the standard MIB (Management Information Base) at each node.*

## 1 Introduction

A network can continue to function, at least for a period of time, without the management subsystem. However, what was a once highly tuned network may gradually degenerate to an inefficient state. Not only a software/hardware failure but also a 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, and trigger the control actions when necessary. As the management tasks rely on the network's status information, a network management system must to 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? Do the format and 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? 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 the problems similar to the information sharing problems in a traditional file system with multiple applications where the application must know the structure of the files it is operating on. 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 the *data independence* for applications [1]. 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 the information sharing may be among the distributed heterogeneous network entities, these distributed databases and the access protocol have to be standardized.

The open-networking community has settled on a management model that places a MIB (Management Information Base) on each network node and manages these MIB's remotely with application level protocols [2, 3, 4, 5, 6, 7]. Figure 1 shows the basic platform for MIB and CMIP (Common Management Information Protocol). Due to the hierarchical nature of network entities and their sub-entities, both ISO and Internet models organize network management information into a hierarchical structure. ISO even encapsulates this hierarchical model into object-oriented databases which hide the heterogeneity of network entities away from the protocol stacks and management applications.

The adopted architecture solves the problems of *information representation* and *heterogeneity of protocol stacks*, but the problem of *information distribution strategy* still remains. This is one of the two problems, namely *information distribution strategy* and *automatic/adaptive management*, we want to solve in this paper.

On the other hand, although the infrastructure of network management is agreed upon the standard MIBs and

[1]This research was supported in part by a grant from Mitsubishi Electric.

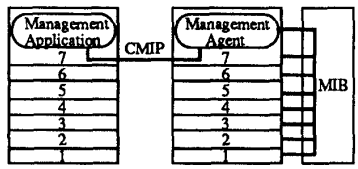



Figure 1: OSI Management Model

CMIP, little was done on 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 models, to cope with fault localization and correction [8, 9, 10]. 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 then the healing procedures for these types of faults. The effectiveness of these systems depends heavily on encoding the problem-solving knowledge in network domain.

Other network management problems also need automation. The maintenance of a large number of objects in MIBs for sure needs to done automatically to keep the status information up-to-date. Configuration management applications can then easily identify and update objects, which in turn changes configurations 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*. That is, the management applications interpret the measurements according to the patterns within the managed network. The ultimate goal for network management should be a self-managed and self-adjustable network with automatic monitoring, problem diagnosis, information interpretation, and control actions.

In this paper, we propose a framework for the 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 pre-specified knowledge 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. MKB (Management Knowledge Base) which includes network patterns, abstract view definition, and control knowledge is represented as a set of logical rules and triggered by the facts in databases and queries from management applications.

Section 2 presents the network management framework for learning and inference. The methodology to build the management information infrastructure is detailed in section 3. Section 4 illustrates the functionalities of MKB in diagnosis, abstraction, prediction, and control.

# 2 The Framework

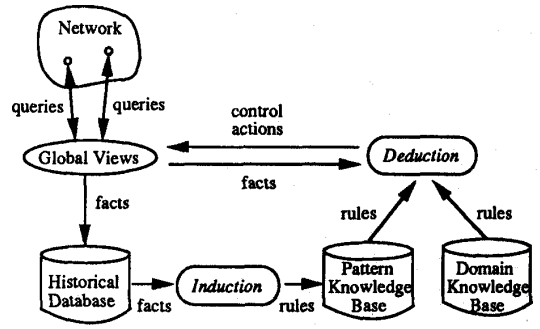



Figure 2: Induction/Deduction in Network Management

To solve the above network management problems: information distribution strategy and automatic/adaptive management, we 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. Network patterns are learned from the measurements stored in a historical database. These discovered patterns, represented in the forms of logical rules, describe the correlation between network objects. Based on these network patterns and pre-specified domain knowledge, forward and backward inference can be triggered to predict network status, fire control actions, diagnose reported problems, and access global views. Figure 2 illustrates the general approach using learning and inference in network management. Unlike an expert system with only pre-specified domain knowledge, the proposed management system has, in addition, learning ability to augment its knowledge regarding the specific managed network. A general approach for fault management system without learning ability is shown in Figure 3 where the usual application is problem diagnosis triggered by forward or backward inference [8, 9, 10].

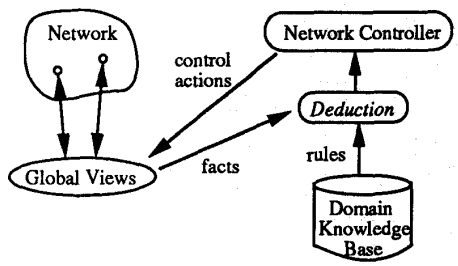



Figure 3: Fault Management with Expert System

Figure 4 is an abstract data flow model of our management systems. EDBs (Extensional Databases) are actually the standard object-oriented MIBs. They represent the basic

facts or data 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 EDBs with logical rules. That is, IDB contains virtual objects defined on the physical objects in EDBs. Access to IDB will be transformed into access to EDBs. 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 [1]. The difference is that now IDB is defined on *distributed* EDBs. IDB, including overall configuration and inter-object relationships, embodies the global views of the network. Extracted from IDB, HDB (Historical Database) is the temporal historical database which encode time in the network trace. Network patterns are learned from HDB and stored in PKB (Pattern Knowledge Base). DKB (Domain Knowledge Base) is pre-specified problem solving and general relationship knowledge. Note that only EDBs are standardized; all the others are management application dependent.

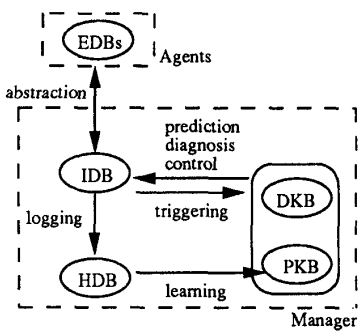



Figure 4: Abstract Model of Information Flow

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 formulas which can represent the status of a network object or an action to update an object's status.

Each network pattern 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 associated with each logical rules shows how strong this pattern is.

If the status of network objects satisfies the body part in the rule, the pattern tell us, from the past experience, 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 a trouble is reported to the management system (eg. blocking probability of connection382 is larger than 5%), this object associated with the trouble is matched with 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 body can carry on. Finally, the set of residual formulas which can not 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.

# 3 Management Information Infrastructure

Modeling network management information is to map network configuration, performance, and events to objects in EDBs. The *inheritance hierarchy* in Figure 5 presents a simple classification of network object classes where *elements* class has three subclasses: *configurations*, *performances*, and *events*. *physical entities* has two subclasses: *nodes* and *links*. Etc.

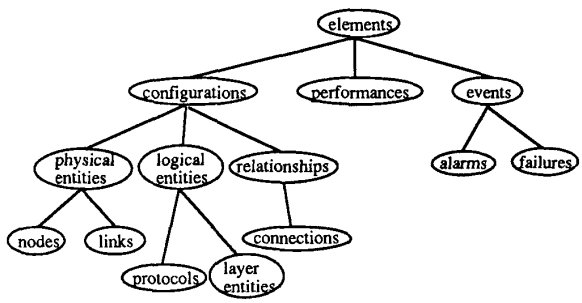
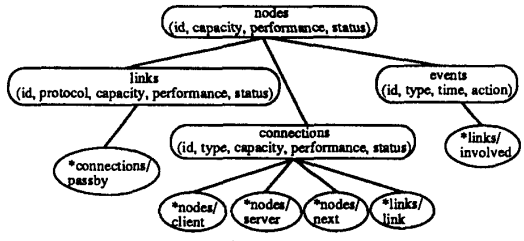



Figure 5: Inheritance Hierarchy

A node's EDB contains only its *local* management information. Figure 6 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 connections that pass 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 this link).

At the management site, what the management applications see is a set of *views* in IDB. Different sets of views can be defined for different management applications. Each view is defined on EDBs with a set of logical rules. The schema at the management site for IDB/EDB and the Prolog implementation to define these views are given in Figure 7. Prolog Logic Programming techniques used here can be found in [11].

To see how a global view can be constructed by combining local MIBs, let us take predicate *connections* as an example. For every "Nodeid", *l_connections*(Nodeid, Connid, Type, Capacity, Perfid, Status, Clientid, Serverid, Nextid, Linkid) contains all connections that pass node "Nodeid". For every such connection, *l_connections* contains "Nextid" and "Linkid" for its next hop (node and link), but doesn't know the whole path. *connections*, constructed




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);

Figure 6: 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).
```

Figure 7: Views in IDB

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 tried to match with the attributes of *connections*. The 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 here 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 EDBs at the management site, and then transferred, by CMIP queries, to the physical EDBs on local nodes. Thus, a mapping between access to predicates of EDBs at the management site and CMIP queries to physical EDBs must be done at the management site. The attribute "Nodeid" in each EDB predicate is used to identify the local node that contains the object instances.

# 4 Management Knowledge Infrastructure

Conceptually, MIB and MKB are organized in the following hierarchical layer structure where EDB is MIB and IDB/DKB/PKB compose MKB:

| Layer | Contained in |
|---|---|
| Control Strategy | DKB |
| Management Knowledge: (Configuration, Performance, Fault) | DKB and PKB |
| Object and View Manipulation Rules | IDB |
| Network Objects | EDB |

The control strategy, which is implemented as the manager, decides when to invoke the submanagers, which actually are rule sets in configuration, performance, and fault domains. 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.

As previously mentioned, both preventive and remedial control actions can be taken by network management applications. Preventive control is triggered by problem forcasting 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 pass 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 occur, 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 pattern rule (in PKB), given that the conditions are true, forecast that the rule goal will be true.
- Control: The forward inference on a domain control rule (in DKB) suggests the control actions to take when some network phenomena are detected.
- Diagnosis: The backward inference on a domain/pattern rule (in DKB or PKB) can discover the root causes of network events, even when they are not yet detected.
- Abstraction: The backward inference on a view definition rule (in IDB) transforms an IDB query to EDB query/queries and hence provides global 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 (ie. 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 condition in rules 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.

# 5 Conclusions and Future Work

In this paper we have identified the basic network management issues: management information infrastructure and automatic/adaptive management. The evolution of MIB and CMIP is briefed. One basic problem in management information infrastructure is to provide a *window* through which management applications can access the *global* management information. Our solution to this is to construct *views* by logic programming on the distributed physical MIBs. Accesses to views are transformed into CMIP queries to MIBs.

We have also proposed the methodology to equip the system with automatic and adaptive management abilities. Learned pattern knowledge works together with domain knowledge to perform adaptive management tasks – prediction, diagnosis, and control action. This pattern knowledge captures the underlying network patterns and refine the prespecified domain knowledge.

An experiment on traffic pattern discovery is reported in [12]. Traffic patterns are learned by a machine learning tool from 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, and predictability. These patterns are useful for medium-term and long-term performance management. Presently, the implementation of a prototype network management system, LEN (Learning Expert for Networks), is in progress.

## Acknowledgements

The authors would like to acknowledge their colleagues, Shen-Tzay Huang and Carlo Zaniolo at UCLA, for the fruitful discussions on issues in deductive databases and logic programming.

# References

[1] Ullman, J. D., *Principles of Database and Knowledge-base Systems, Volume I,* p. 11-12, p. 82-87, p. 100-101, Computer Science Press, 1988.

[2] Case, J. D., J. R. Davinm, M. S. Fedor, and M. L. Schoffstall, *A Simple Network Management Protocol,* RFC 1067, SRI Int., August 1988.

[3] McCloghrie, K. and M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets,* RFC 1156, Internet Standard, May 1990.

[4] Rose, M., *Management Information Base for Network Management of TCP/IP-based Internets - MIB II,* RFC 1158, Internet Standard, May 1990.

[5] ISO/IEC DIS 10165-1, *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 1: Management Information Model,* ISO, Geneva, Switzerland, June 1990.

[6] ISO 9596, *Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification,* ISO, Geneva, Switzerland, May 1990.

[7] Cassel, L. N., C. Partridge, and J. Westcott, *Network Management Architectures and Protocols: Problems and Approaches,* IEEE Journal on Selected Areas in Communications, Vol. 7, No. 7, September 1989.

[8] Liebowitz, J. (Editor), *Expert Systems Applications to Telecommunications,* John Wiley & Sons, New York, 1988.

[9] Ericson, E. C., L.T. Ericson, and D. Minoli, editors, *Expert Systems Applications in Integrated Network Management,* Artech House, 1989.

[10] Goyal, S., *Knowledge Technologies for Evolving Networks,* Proceedings of the IFIP TC6/WG6.6 Second International Symposium on Integrated Network Management, January 1991; also in Integrated Network Management, II, I. Krishnan, et al., editors, North-Holland, 1991.

[11] Sterling, L. and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques,* MIT Press, 1986.

[12] Gerla, M. and Y. D. Lin, *Network Management Using Database Discovery Tools,* Proceedings of IEEE 16th Conference on Local Computer Networks, Minneapolis, October 1991.