Chapter 4

Fault Management

Today’s high-speed heterogeneous networks represent complex and data intensive environments demanding higher levels of human manager expertise and involving increasingly complex overhead. Moreover, the networks themselves are also continuously evolving due to the competitive nature of the industry and continuing technological improvements. Usually, these networks are gigantic and multifaceted due to the diversity of different types of communications equipment; even the error message volume is so large that a well skilled operator finds it difficult to handle all faults of the network. That is why fault management becomes one of the key functions of integrated network management systems.

Network problems can be classified as hardware and software faults that cause elements of the network to behave incorrectly. Fault constitutes a class of network events that can cause other events but are not themselves caused by other events. Faults may be classified according to their duration time as: (1) permanent, (2) intermittent, and (3) transient. A permanent fault exists in a network until a repair action is taken. Intermittent faults occur on a discontinuous and periodic basis causing a degradation of service for short periods of time. However, frequently re-occurring intermittent faults significantly jeopardize service performance. Transient faults cause a temporary and minor degradation of service. They are usually automatically repaired by error recovery procedures. The primary purpose of fault management to identify, trace and solve network problems. Typically, the flow of fault management activity is as follows:

- Collect messages from network components;
- Generate alarms by filtering the messages;
- Diagnose the faults that caused the messages by correlating the alarms;
- Determine a plan for correction of faults;
- Correct the fault; and
- Verify that the network problem is eliminated.
These activities imply a number of decisions such as when and where to send a network manager, whom to send (with what qualifications), and which equipment is required in order to restore the normal operation of the network. These decisions must be made either by network administrators or by some intelligent component that must be integrated with the fault management system. The fault management process consists of three stages as depicted in Fig. 1. The first stage is fault detection, the recognition that a problem exists in the network. Fault isolation processes this fault information to determine the primary cause of the detected signals or alarms. This stage then sends its results to fault correction, where the primary cause of the network problem is corrected. From this point of view, one can consider fault detection as the "front-end" to the fault diagnosis stage. Similarly, fault correction can be viewed as the "back-end" to fault diagnosis.

![Fig 4.1 Fault Management Process](image)

1. Fault Detection

Fault management begins with fault detection, that is, the determination that a problem exists in the network. Within fault detection, the network is monitored for anomalous conditions, typically through a combination of automated and non-automated procedures. The data obtained during fault detection is collected from the network management data stream. For example, in Fig. 1, the information coming from the log of Alarms contains status messages as well as error messages. Fault detection data may also contain messages from an ancillary network within the overall telecommunications system. For example, contact closure monitors can transmit their messages on a network dedicated to network monitoring and management. When polled, these devices report "on/off" settings. The data and messages are converted to a form suitable for subsequent processing and may also be stored for subsequent fault administration purposes.
2. **Fault Isolation**

Fault isolation is a central aspect of network fault management. Since faults are unavoidable in communication systems, their quick detection and isolation is essential for the robustness, reliability, and accessibility of a system. In large and complex communication networks, automating fault diagnosis is critical. Once a network anomaly has been detected, the next task is to accurately identify its cause. The cause for network failure determined in this stage includes those network elements which are generating the diagnostic messages, for example, a switch failure. Ideally, the diagnosis also includes the primary cause of a failure, such as peak load at a switch. This information is then passed to the fault correction stage, where solutions are specified. Key issues in fault isolation include identifying a primary or most likely cause and determining the most urgent case requiring attention. In the past, numerous paradigms were proposed upon which fault localization techniques were based. These paradigms derive from different areas of computer science, including artificial intelligence, graph theory, neural networks, information theory, and automata theory. In Fig. 2, a classification of the existing solutions is presented. These solutions include techniques derived from the field of artificial intelligence (rule-., model-., and case-based reasoning tools as well as decision trees, and neural networks) model-traversing techniques, graph-theoretic techniques, and the codebook approach.

![Fig. 4.2. Classification of fault isolation techniques.](image)
Artificial Intelligence based approaches for fault isolation

The most widely used techniques in the field of fault isolation are expert systems. Expert systems try to reflect actions of a human expert when solving troubles in a particular domain. Their knowledge base imitates knowledge of a human, which may be either surface resulting from experience, or deep resulting from understanding the system behavior from its principles. Most expert systems use rule-based representation of their knowledge-base. In the domain of fault localization, the inference engine usually uses a forward-chaining inferencing mechanism, which executes in a sequence of rule-firing cycles. In each cycle the system chooses rules for execution, whose antecedents (conditions) match the content of the working memory. Expert systems applied to the fault localization problem differ with respect to the structure of the knowledge they use. Approaches that rely solely on surface knowledge are referred to as rule-based reasoning systems. The core knowledge may be understood as a generic or reusable knowledge. It is useful to identify an approximate location of a fault in a large network. Customized knowledge allows us to precisely isolate a fault from the selected group of system entities and composite events, which are composed of primitive and other composite events, and include a set of conditions that have to be verified before a composite event can be asserted.

Rule based systems rely solely on surface knowledge, do not require profound understanding of the underlying system architectural and operational principles and, for small systems, may provide a powerful tool for eliminating the least likely hypotheses. Regardless of the type of knowledge used by expert systems, the fault localization process is driven by an inference engine according to event-correlation rules, which are usually defined using a high-level language. Rule conditions are frequently expressed as patterns, which test alarm frequency and origin as well as values of alarm attributes.

Case based systems are a special class of expert systems that base their decisions on experience and past situations. They try to acquire relevant knowledge of past cases and previously used solutions in order to propose solutions for new problems. They are well suited to learning correlation patterns. When a problem is successfully solved, the solution (or its parts) may be used in dealing with subsequent problems. They
are resilient to changes in network configuration. However, case-based systems require an application-specific model for the resolution process.

In addition, decision trees and neural networks have been used for the purpose of fault localization. Decision trees are used as a representation of an expert knowledge to guide a user observing symptoms of failure toward locating the root cause of the problem. Neural networks are systems composed of interconnected nodes called neurons, try to mimic operation of a human brain. They are capable of learning and resilient to noise or inconsistencies in the input data.

Model traversing techniques use formal representation of a communication system with clearly marked relationships among network entities. By exploring these relationships, starting from the network entity that reported an alarm, the fault identification process is able to determine which alarms are correlated and locate faulty network elements. Event correlation in model-traversal techniques is usually event-driven. For every observed event, which is typically mapped into a model node, the model is searched recursively following the relationship links between managed objects. Model traversing techniques are robust against frequent network configuration changes.

Graph-theoretic techniques require a priori specification of how a failure condition or alarm in one component is related to failure conditions or alarms in other components. To create such a model, an accurate knowledge of current dependencies among abstract and physical system components is required. The efficiency and accuracy of the fault localization algorithm are dependent on the accuracy of this a priori specification. Fault propagation models take the form of causality or dependency graphs.

The divide and conquer algorithm uses a dependency graph. The algorithm first identifies the alarm domain, a set of all faults that may have caused the observed alarms. In the first phase, the alarm cluster domain is partitioned into two subsets characterized by the maximum mutual dependency. In the second phase, the algorithm decides which of the subsets should be used to recursively invoke the fault localization procedure. If the subset with higher probability that one of its members was a primary source of a failure is able to explain all alarms, the recursive procedure is invoked using the entire alarm cluster passed to this recursion step and the higher-probability subset as
parameters

The natural feature of **context-free grammars**, which allows expressions to be built from sub expressions, may be effectively used to represent a hierarchically organized communication system. In this model, terminals correspond to the indivisible network components terminal objects. Productions are used to build compound network objects. Fault localization problem using context-free grammars as a fault propagation model has addressed the problem of locating the source of failure. The fault localization problem using an unrestricted context-free grammar model can also be formulated as an integer non-linear programming problem.

The **codebook** is basically a matrix representation of a bipartite causality graph that has been optimized to minimize the number of symptoms that have to be analyzed while still ensuring that the symptom patterns corresponding to problems allow us to distinguish between the problems. Fault propagation patterns in the codebook technique are represented by a matrix of problem codes that distinguish problems from one another. In the deterministic technique, a code is a sequence of values from {0, 1}. The value of 1 at the i\(^{th}\) position of a code generated for problem \(p_i\) indicates cause-effect implication between problem \(p_j\) and symptom \(s_i\). In the probabilistic model, codes contain values [0, 1] that represent likelihood of the cause-effect implication between a given problem and a given symptom.

3. **Fault Correction.**

The main input to fault correction is the categorization of the primary cause of the network fault. Fault correction then formulates a set of feasible solutions to the network problem. As with fault detection, this stage typically consists of a mixture of automated and non-automated activities. Ideally, a "best" solution (based upon \(a \text{ priori}\) criteria) is chosen from the feasible solutions and subsequently implemented. Human judgment is typically desirable to resolve final solutions and actions may be required from network personnel to implement a specified solution. Therefore, diagnostic information and recommendations must be presented in a format that facilitates human information processing.
Corrective actions may be automated in various cases, such as redirecting traffic from congested network elements. Here, based upon diagnostic messages or requests from personnel, predefined fault correction command sequences are used to restore services. For example, the network management organization responsible for the switch settings typically has several alternate tables of such settings. When a previously analyzed case arises, messages are sent to the switches to use the appropriate routing table. A key goal of the fault correction stage is to learn and capture human judgment. Automation is being introduced in this stage, as in the previous stages, but the degree of automatic network restoration may depend on the problem domain, the severity of the problem, or the intelligence of the system components.

Overall management of data and procedures required in fault management is controlled by the fault administration component. Activities include monitoring and facilitating the interaction among the various stages and components of the fault management system. In addition, system performance may be monitored, and improvements proposed. The fault history database may also be part of this component. Because of the diversity of data and knowledge sources in fault management, fault administration should allow system administrators, domain experts and network engineers appropriate shared access to network information to concurrently share associated resources as needed. The automation of management activities can minimize the human involvement, reducing management errors, and, in consequence, lowering the cost of network operation and maintenance. The increased productivity improves the performance of the entire network and minimizes network outages.

**Mobile Agent based Automated Fault Management**

The basic idea to resolve network faults is to bring management intelligence as close as possible to the managed resources. One of the most prominent techniques providing a solution is Management by Delegation (MbD). It represents a clear endeavor towards decentralization and increased flexibility of management functionality. Instead of the traditional methods of exchanging client/server messages, the management station can specify a task to be carried out by locating an agent on involved devices, where the actual execution of the task takes place. Such execution is completely asynchronous, producing
a higher degree of parallelism, and thus enabling the management station to perform other tasks. The management station action of delegating a specific function to a remote process is described as function delegation.

The second property that enhances the functionality of the mobile agent is the ability to travel from one node to another. This capability is termed mobility. Code mobility is defined to be "the capability to reconfigure dynamically, at run-time, the binding among software components of applications and their physical location within a computer network". The third capability from which fault management systems can benefit is the integration of Artificial Intelligence technologies, which capture the human manager expertise in solving network problems. Thus, the management station delegate tasks to autonomous intelligent code that is capable of making independent decisions on the manager's behalf. The decisions are based on the data that the code analyzes locally at the hosting nodes without human manager involvement. Combining these three capabilities (i.e., agency, mobility, and intelligence) is a software entity coined an **Intelligent Mobile Agent**. Through the integration of the three properties, intelligent mobile agents improve network manageability by dynamic distribution of management intelligence to the networked devices. Instead of querying each node to gather the information needed to perform fault management, the management station generates a mobile agent and then dispatches it to the network.

![Fig. 4.3 Schema of intelligent mobile agent.](image-url)
The agent's goal is to visit all of the nodes that contain any information relevant to the assigned task. The agent processes the information locally, draws conclusions, undertakes appropriate actions, and either communicates the findings to the dispatcher or carries the results away as it leaves this node. What the agent does is termed as a semantic compression of the network component data, which produces a set of hypotheses about the network status. When the agent returns to the network manager, it presents the supervisor with the findings (if it has not done it already). Moreover, relying on the intelligence embedded in the agent, the management station may allow an agent to exercise more freedom.

Mobile Agent Based Intelligent System Architecture

![Architecture of Mobile Agent based Intelligent System for Fault Management.](image)

The Fig 4.4 describes the architecture of the mobile agent based intelligent system for automated fault management solution. As depicted in the figure, the network to be monitored is the collection of network elements such as routers, switches, end-user devices.

**PMA (Platform for Mobile Agent)** is the platform for the development and the management of mobile agents that gives all the primitives needed for their creation, execution, communication, migration, etc. It has been entirely developed with IBM Aglets. The IBM aglets provide JAVA API for programming mobile agents and an environment for
Mobile Agents based Intelligent System for Autonomic Network Management

running mobile agents in Java. Being developed with Java, it ensures its total portability on the different hardware and software architectures. One of our purposes in the creation of the platform has been the complete integration and compatibility with the SNMP world. For this reason, in our system we have integrated and used some classes created by WebNMS, which implement the SNMP stack. In this way, we had the opportunity to interact with the different nodes of the network through standard MIB variables. At the same time the developed framework allows us to monitor non-standard quantities (not defined by a MIB) defined by the user. The various mobile agents used for fault management process along with their detailed functionality are described as under:-

The **browser agent** collects some MIB variables from a set of nodes specified by the user. After being started, the agent reaches the first node to be visited, opens an SNMP local communication session, builds a PDU-request containing the MIB variables to be searched, waits for the reply from the SNMP daemon, and saves the data obtained in an internal structure. Then, if other nodes need to be visited, it reaches them, and repeats the procedure mentioned above. This agent realizes the functionalities of an extended MIB browser. In fact, in an ordinary MIB browser, after activating a connection with a specific node, we can view the contents of the MIB variables, by sending an SNMP request to the node in question for each variable. Conversely, through the browser agent, such requests are first given to the agent, which (by moving from a node to another) inter-acts with them locally and reports the global result to the initial station.

The **daemon agent** monitors a “health function” defined by the user. For starting the agent, the function to be computed and the node of the network (where the computation has to be done) must be provided. Then this agent moves to the node in question, where it records the value of the function: if the value is higher than a fixed threshold (defined by the user), a notification message is sent to the server from which the agent has departed. We can implement (by means of this agent) a mechanism of generalized trap, where the events in which the NMS is notified can be freely and very flexibly set by the user.

The two agents described before directly interact with the SNMP daemon present in the different nodes (through the Advent classes). Conversely, the **messenger agent**, during its migration through the nodes of the network, interacts with other agents for collecting
specific information produced by them. During the configuration we need to select the agents to be contacted and the servers where they have to be searched, and (if necessary) also the number of times the agent has to contact such agents. Thus, the messenger agent performs operations at a higher abstraction level than the mere retrieval of MIB variables. In fact, since daemon agents can perform the computation of any function on the different nodes of the network, the messenger allows us to collect such information, thus obtaining a general description about the state of the network.

The **verifier agent** does not perform an actual Network Management action. Its task is that of collecting important information, which might be useful for further operations of Network Management, by MAP. It visits the nodes selected during the configuration, and computes a function whose purpose is the evaluation of the presence of a specific property of the system visited (for example, we might think of the verification of a software version, or the available space on disk not below a fixed threshold, or the verification of some log files, etc.). The verifier agent then reports the server, from which it departed, the list of the nodes that satisfy this property.

All the coming events are captured by browser agent and are stored in event and topology databases. The Control Agent verifies integrity of databases and used them with Event Correlation Logic to correlate the coming events with the appropriate faults. When the fault is localized, these correlated alarms/events are sent to the Inference Engine of the expert system. Being the core component of the proposed network management solution, the captured correlated alarms from PMA are inferred by **Inference Engine** of Expert System module. Inference mechanisms are control strategies or search techniques, which search through the knowledge base to arrive at decisions. As expert systems predominantly process symbols, the inference process manipulates symbols by selection of rules, matching the symbols of facts and then firing the rules to establish new facts. This process is continued like a chain until a specified goal is arrived at. In an expert system, inference can be done in a number of ways. The two popular methods of inference are backward chaining and forward chaining. Backward chaining is a goal-driven process. it tries to establish goals in the order in which they appear in the knowledge base. the inference process will stop once goal variable gets a value. Forward chaining is a data-driven inference process. The user of the system has to give all the available data before the start of the inference. The
inference mechanism tries to establish the facts as they appear in the knowledge base until the goal is established. The inference process selects the first rule in the rule base and discards it if the first condition itself evaluates to false. Then it goes to the second rule, and if it evaluates to true and it is fired, resulting in a new fact being added to the working memory and the cycle repeats.

In large rule bases, a number of iterations may be required before the goal is established. The order in which rules appear in the rule base plays a major role in the way inference is carried out in forward chaining, whereas such order does not play any role in backward chaining. But the order in which conditions are listed in a rule is important in backward chaining. The order in which questions are asked to the user for response depends on this order. Hence, before formulating the rule base, the knowledge engineer should decide whether backward chaining or forward chaining is going to be adopted for reasoning. The explanation facility of the expert system provides a mechanism for how a fact is established. The answer requires reference to the knowledge base. The process associated with the query searches for the rule with the required fact in the then part and displays the rule, which shows how the fact is established.

The knowledge base contains the domain-specific knowledge required to solve the problem. The knowledge base is created by the knowledge engineer, who conducts a series of interviews with the domain expert and organizes the knowledge in a form that can be directly used by the system. The knowledge engineer must have the familiarity of knowledge base expert system technology and should know how to develop an expert system using a development environment or an expert system development shell. It is not necessary that the knowledge engineer be proficient in the domain in which the expert system is being developed, but a general knowledge and familiarity with the key terms used in the domain is always desirable, since this will not only help in better understanding the domain knowledge but will also reduce the communication gap between the knowledge engineer and the expert. Before deciding on the structure of the knowledge base, the knowledge engineer should have a clear idea of different knowledge representation schemes and the suitability of each under different circumstances.
The knowledge that goes into problem solving in engineering can be broadly classified into three categories, viz., compiled knowledge, qualitative knowledge and quantitative knowledge. Knowledge resulting from the experience of experts in a domain, knowledge gathered from handbooks, old records, standard specifications etc. forms compiled knowledge. Qualitative knowledge consists of rules of thumb, approximate theories, causal models of processes and common sense. Quantitative knowledge deals with techniques based on mathematical theories, numerical techniques etc. compiled as well as qualitative knowledge can be further classified into two broad categories, viz., declarative knowledge and procedural knowledge. Declarative knowledge deals with knowledge on physical properties of the problem domain, whereas procedural knowledge deals with problem-solving techniques.

The **Knowledge Acquisition Process** (KAP) initiated by expert system module is used to build the diagnosis/corrective actions as part of expert system module case library. The domain expertise and the vendor specific information are sources of input for KAP. Human network expert advice in from of corrective actions is also incorporated by KAP. The learning module in between expert system module case library and KAP constantly refines the expert system module case library by building the new corrective actions to be taken for the new event. Rule base library as well as the working database for finding the associated corrective action which should be initiated to resolve the network fault and if the corrective action is automated it executes the necessary scripts in the form of corrective action.

The expert system module will analyze each scenario presented by the monitor, identify various problem symptoms, diagnose faults, isolate probable causes, and generate solutions. We have implemented rule based expert system. The core component of expert system is the knowledge base where the “expertise” of the system is stored. The knowledge base is represented as a set of rules. Rules are “if-then” structures. If certain criteria are met, then the system is to take certain action. The expert system’s problem solving capabilities are directly dependent on the number of rules it has in its rule-base. By increasing the number of these rules, the expert system will be able to solve many and more complicated problems.
The **working database** contains the rules that are directly applicable to the given problem under consideration. These are the rules that need to be “fired” – executed since their “if” clauses have been satisfied. The system updates the working memory by asserting, modifying, or retracting working memory elements. For a network application, the working memory typically contains a representation of characteristics of the network related to the current problem, including topological and state information. The rule-base expert system represents knowledge about what operations to perform when the network enters an undesirable state. The expert system module employs heuristic techniques for finding correct responsive corrective action for faults which includes forward chaining and backward chaining. In case of forward chaining, the hypothesis is taken first and the rules are verified to prove the hypothesis whereas the process reverses in case of backward chaining. Given a symptom (or a set of symptoms), the heuristics initially associate the symptom(s) with particular corrective action and assert strategy (and/or tests) to adopt in diagnosis the system event correlation strategy is framed on the following patterns

If < symptom-patterns >
then < diagnosis-class>

Here, the <symptom-pattern> depicts heuristic rule specifying various network conditions and <diagnosis-class> is hypothesis showing the corrective action to be taken to troubleshoot the network. the inference engine evaluates the hypothesis to be true if the corresponding rules evaluate to true. the following examples depict the specific network conditions as SNMP traps and system log messages and the corrective actions executed by expert system module.

**Example # 1**

Consider an environment with Cisco router 7600 and NMS station gets the following System Log (Syslog) message:

ERROR: SNMPTRAP (Chassis Alarm)

**Problem Description**: The operating Temperature of the networking device exceeds the threshold limit.

**Corrective Action**:

If String (Message (Chassis Alarm))
then Shut_Router (R)

The above stated Corrective Action is applied to shut down networking device Router ( R )
Example # 2
Consider Cisco 7500 router, a Cisco 5000 Catalyst Switch in a LAN Environment and the following syslog messages are logged by NMS:
Error: SNMPTRAP (E0_Router, bulk_traffic)

**Problem Description:** The Interface E0 of Router experiencing bulk traffic i.e. no. of packets entering the interface exceeded threshold limit.

**Corrective Action:**
If String (Message ((Snmptrap_Bulktraffic)))
Then Router(R, Increase_QueueLength) AND Switch(S, inc_time_out, inc_retry)
The correlation rule stated above is applied to try and alleviate the congestion conditions by increasing the router queue length, Timeout and Retry values for Packets sent over the Router and Switch Interface.

In case of a “new” event, it is piled up to the trouble ticketing system. all new events are assigned ticket id by the ticket id generator module and are stored in trouble ticketing database. the events are categorized on basis of priority by priority control logic as some events require immediate attention than others. all such events are compiled by experts/ network administrator who initiates sequence of corrective actions for a particular event this human network expert advice in from of corrective actions is piled in knowledge database by KAP.

**Implementation**
A prototype implementation of expert system module is available and has been tested for a variety of scenarios on PMA, while a proof-of-concept Helpdesk / Trouble Ticketing System was built for testing purposes. Figure 2 depicts the logical working of expert system module while troubleshooting network conditions to generate the corrective actions to be taken. Only critical Alarms from PMA were configured to be forwarded to the helpdesk utility, which assigned a unique trouble-ticket ID to the incoming critical event (internally designated as a “call” in helpdesk parlance) and assigned an “expert” network administrator from a pre-configured list to the “call”.
The helpdesk utility also classified the “calls” into various severity levels by the use of priority control logic. For instance, network infrastructure related calls (routers, switches, or critical servers) are assigned the highest priority whereas individual host related calls are assigned the lowest priority. The helpdesk utility then provides an easy-to-use interface to the network administrator to record the steps taken in the diagnosis and resolution of the reported fault serving as an input to expert system module. Expert system module basically works on three inputs:

1. Correlated alarms from the PMA which provide the relationship between various event patterns and help Expert system module in learning about the various symptoms of a reported network fault.
2. The expert action taken by the network administrator in diagnosing and resolving critical network problem
3. Static pre-configured rules describing the relationship between device specific faults and recommended corrective action by the device vendor (wherever such information is available)

The SNMPTRAP utility was utilized to generate specific events and event patterns for testing purposes, which were correlated by PMA and forwarded to the helpdesk utility. The Static pre-configured rules for the Expert system module system were adapted from the Cisco Event Correlation Guidelines, which is a detailed document published by Cisco Inc. describing detailed event correlation scenarios for its devices and the causes for such events.

Expert system module incorporates the logic for decoding the event stream from PMA. For this purpose, it makes use of the event APIs to enable it to parse the incoming events and extract useful information from it such as event name, event OID (object identifier), event source, event severity etc. This information includes the SNMP community strings needed by E-Net to execute any SNMP-SET operations on network
devices as part of the automated corrective actions:

REFERENCES

networks, 57-60.