Model Based Diagnosis in LANs

Sandro T. Fontanini[1], Jacques Wainer[2], Volnys Bernal[3], Silvio Maragon[4]
[1] Support Analyst, Ericsson, Brazil, email: sandro.fontanini@edb.ericsson.se Phone: +55 11 6224-8934
[2] Instituto de Computação, UNICAMP, Brazil e-mail: wainer@ic.unicamp.br Phone: +55 19 37885871
[3] LSI, USP, Brazil e-mail: volnys@lsi.usp.br Phone: +55 11 3818-5675
[4] LSI, USP, Brazil e-mail: maragon@lsi.usp.br Phone: +55 11 3818-5675

Abstract— **This paper presents a model-based approach to diagnose performance and communication faults in local area networks (LANs). The diagnostic is based on the idea that if the system has an approximate model of the network, and knows how IP packages travels in the network, it can reason about the causes of alarms. A set of subsystems was developed, to gather the network information, to construct the model, to criticize the model, and to interact with the network in order to gather status information. Some results of experiments are reported.**

Keywords— **communications faults, performance diagnostics, model based reasoning**

I. Introduction

Many of the Artificial Intelligence diagnostic systems for computer network faults discussed in the literature (for example [1], [2], [3], [4], [5], [6], [7]) or available as part of network management platforms are *experience-based* systems. Such systems, in some way capture the experience of detecting a fault on a particular computer network. For example, [1], [2] discuss case based models for fault diagnostics: a memory of previous faults episodes of the network is kept, and as a new set of abnormal behaviors are discovered, they are matched with the previous episodes, in the hope that previous knowledge will help solve the current problem. Other systems use neural nets [8] to learn from previous cases to diagnose a new fault.

A problem with experience based systems is that they only work for that particular network. If a particular network, for which one has a large set of diagnosis cases, is changed, it is very likely that a case based system, or a neural net learning system, and any other experience based system will be useless. The experience based system is tuned to that particular static network (topology, type of equipments, even identity of the equipments) and any changes in the network renders the system useless. It is our experience that local area network undergo changes with high frequency (one change a week), and thus the need for diagnostic systems that are not so dependent on the particular network.

This paper will present a model-based approach to detection of performance degradation and fault diagnosis in computer networks. The idea is to have an approximate description of the local area network: the topology, type of equipments, routing tables, and so on. With this information (and knowing of how IP packets travel in the network) one is able to perform a model based reasoning for this network. For example, let us suppose that the management platform detects that a host is unreachable. By computing the path that the IP packet would have taken to and from that machine, one can create a first diagnostic hypothesis, that at least one element in the path is faulty. By either querying further the network or receiving further alarms the hypothesis is incrementally refined. When the hypothesis can no longer be refined, it is informed to the (human) network manager, to further refine the hypothesis but in locus observation, or to correct the network by fixing the components.

To illustrate the diagnosis of a performance degradation situation, let us assume that the diagnostic system discovers that the latency for a IP packet to reach a certain destination and come back is too high. By computing the path the packet takes, and assuming that the high latency is due to a segment of the path with high occupancy of the channel, the system can create a first diagnosis hypothesis, that one of the collision domains in the path has a high occupancy rate. A *collision domain* is a set of devices interface connected with some technology so that an unicast packet sent by any interface in the domain is also received by all other interfaces in the domain; thus all interfaces connected among themselves through HUBs, or an Ethernet cable are in the same collision domain. By further investigations of each of the collision domains in the path, the system can refine the hypothesis and determine the particular collision domain which has high occupancy rate and which hosts are responsible for it.

In order to construct the model of the network we developed a module that discovers the network model automatically. Thus, if the network is changed, this network discovery system must be run and it will automatically (or semi-automatically) reconstruct the network model for the new configuration.

A. Architecture of the diagnostic system

The diagnostic system, pictured in figure 1, is composed of different subsystems, which run at different moments. The off-line subsystem, has two components: the Network Discovery System (NDS) and the Configuration Critic System (CCS). These two systems generate and analyze the network model.

The online subsystem has two components: the Network Status Gathering System (NSGS), which collects information about communication faults and performance degradation from the network, and the General Diagnostic System (GDS), the central component of the system, which receives the information of some communication fault or unacceptable latency, and based on the network model, interacts with the NSGS to discover a set of diagnostic hypothesis.

Each diagnostic hypothesis is a list of network elements and connections, where at least one of these elements is faulty.

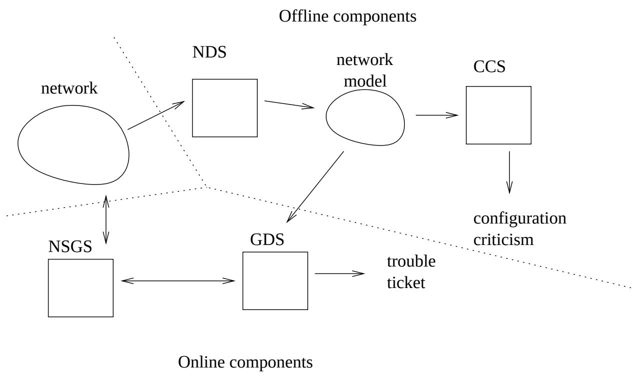

Fig. 1. The diagnostic system online architecture

II. Network Discovery System

In general, commercial network management platforms have a discovery module that is capable of constructing part of the network model, but which usually does not contain all the necessary information needed for a diagnostic system. Such platforms frequently collect information about the network level (IP level) and about the identity and parameters of the equipment; sometimes they also collect information about the connection and interface characteristics, but information about routing tables is usually not gathered by such systems.

In this work we have developed a NDS that can collect such information. First, the NDS sends one broadcast ping (ICMP-echo) for the LAN, listing all SNMP agents that answer for this ping [9]. Making use of this list, the NDS starts to fetch all MIB information found in which agent of the network, organizing this information per sub-network. The NDS makes use of the SNMP, and in particular interfaces and IP group from MIB-II [RFC1213], Bridge MIB [RFC1493], RMON MIB and Repeater MIB [RFC1516]. The following table shows all entries that are collected:

ipRoute	ipRouteDest ipRouteNextHop ipRouteIfIndex ipRouteMask ipRouteMetric1
ipNetToMedia	ipNetToMediaPhysAddress ipNetToMediaNetAddress rptrAddrTrackPortIndex
Rptr	rptrAddrTrackSourceAddrChanges rptrAddrTrackNewLastSrcAddress rptrAddrTrackPortIndex
dot1dTp	dot1dTpFdbAddress dot1dTpFdbPort dot1dTpFdbStatus
ifTable	ifIndex ifPhysAddress
ipAddr	ipAdEntAddr ipAdEntIfIndex
sysname	sysName

TABLE I
Entries from the MIBs

After fetching and organizing this information, the NDS starts one detailed parsing, translating the data into Prolog clauses and building the network model. Basically, the NDS performs the following reasoning when parsing:

- The information from the IpNetToMedia table provides all available IP/MAC pairs (it is also possible to fetch additional pairs in the bridge-mib table). This makes possible to physically identify the devices
- The information from the rptr table belongs to hubs, where it is possible to find all the Collision Domain connections and the up-link connection when intersecting with the IpNetToMedia table
- The dot1dTp table provides information of connection and interfaces that connects the switch with respective devices
- The core information to the diagnostic is fetched from the IpRoute table. The routing tables can be built from them, all route computation consults these tables. Information like next hop, destination IP, source IP are found for each device.
- The interfaces for each device are fetched in the IpAddr and IfTable tables

The model constructed is usually an incomplete model of the network. If there are some network equipments that are not SNMP-managed, then part of network connections cannot be automatically gathered. It is unfortunately very common to find not SNMP-manageable hubs and switches that do not appropriately implement the Bridge MIB. This incomplete network model will lack the information to distinguish the topology within a collision domain, but will have information about all computers, switches and routers in the collision domain. The remaining information has to be entered manually into th NDS model. Although such task is not trivial, the partial model allows for some guidance in collecting such information. Once such information is entered into the model it is assumed to be constant, so in a new cycle of discover, such information is kept except if it contradicts with the newly gathered information. Thus, the manual work of entering the missing connection information must be performed only once.

III. Configuration Critic System (CCS)

The Configuration Critic System performs many tasks. First it analyzes the quality of the model gathered by the NDS, specially in relation to its incompleteness. The CCS evaluates if the possible lack of information regarding physical connections within a collision domain will render the model too imprecise to be useful or not. In particular for the diagnosis of performance degradation the collision domain internal configuration information is not needed. But when dealing with communication faults such lack of information may be serious.

Once the model has been verified for completeness, the CCS will analyze the network itself for contradictory information or for situations considered incorrect. We found out that in a network of hundreds of components it was common to have at least a dozen configuration problems. For example, the CCS will check if all routing tables are consistent with each other, it will verify if there is an IP number assigned to different equipments, if whenever naming is available, it is consistent, and so on. The CCS will also suggest some configuration improvements, like pointing out that two servers are in the same domain of collision and consequently may overload this domain.

IV. The General Diagnostic System (GDS)

A. Network Status Gathering system (NSGS)

The Network Status Gathering System (NSGS) is the component that queries the network to determine its status. Among the information returned by the NSGS there is a "loss of signal" alarm that indicates that the queried device did not answer a ping (ICMP echo requests) and the latency of the ping when it succeeds. The NSGS can also query SNMP objects.

A.1 The NSGS and the polling heuristic

The NSGS uses the LAN model, in a first instance, to sort the elements that will be polled. It is known that the polling of a large network is expensive, both in terms of time and resources, increasing the traffic overhead and decreasing the performance. For this reason, it is important to maximize the efficiency of the pooling. [10] describe a pooling heuristics based on the time interval between ICMP-echo packages sent to the same device. We have developed an heuristic based on the merging of elements of different collision domains, computing the maximum distance between devices of the same domain.

We first build a tree (see figure 2), having as root the manager device (the device that runs this system). It is clear that only the leaves of this tree must be polled. If there is no connection to a leaf, it is assumed that at least one device in the path to and from that leaf is in the faulty state, and such hypothesis is refined by pooling the elements in the path.

One feature of this tree is that one set of sister leaves represents the same collision domain. For this reason it is possible to interactively merge leaves, until no more leaves are found. At the end of this operation we can guarantee that the final merging is an semi-optimal poll sequence where two pings to the same collision domain will be placed in a long distant from each other, although we cannot guarantee optimality (this is an approximate solution for an NP-complete problem if any node has more than two subtrees).

Once the poll sequence is computed (by the CCS), the system starts the polling loop, which is interrupted when either a polled device does not answer to the ICMP-echo or when the latency of such an answer is larger than a predefined threshold. In such cases, the General Diagnostic

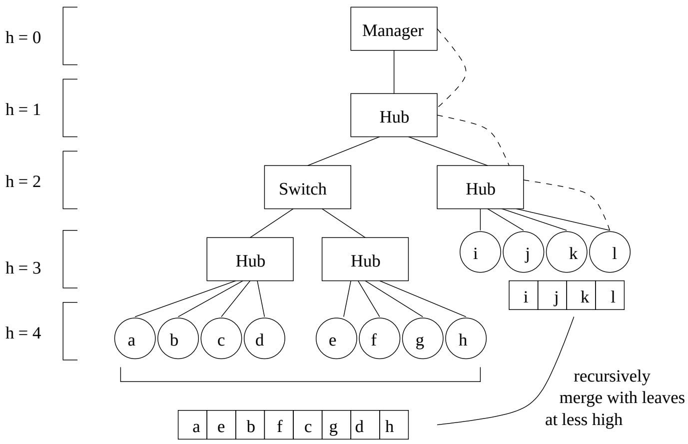

Fig. 2. The network tree

System is started.

V. Fault diagnosis

Upon detecting that one device is not answering the polling, the diagnostic system computes the path the IP packet would have taken from the NSGS to the device. It then assumes that one of the devices in the path is faulty and queries these devices according to different orders, in order to reduce the size of the set of "possible faulty devices".

Our fault detection algorithms make the following assumptions:

- the network (LAN or WAN) uses no dynamic routing protocol.
- TCP/IP based network.
- during the diagnostic cycle there is a single fault in the system.

The first two presuppositions are needed to calculate the path the IP packet took from the NSGS to the first device that was discovered as unreachable. We believe that it is very common for LANs and even certain WANs to have dynamic routing disabled. That is true, at least, in our test network, a departmental research oriented network with hundreds of computers. The subparts of the network that are ATM based are not included in the tests.

The third assumption is reasonable given the speed of the diagnostic cycle. Once a "loss of signal" alarm is received, the diagnostic cycle can in few seconds determine a minimal set of possible faulty devices. In these seconds it is unlikely that some other device will also fault.

A. Generic diagnostic algorithm

The generic diagnostic algorithm is described below. `generic(From,Dest,`H`)` should be interpreted as a function that given that H is the set of possible faulty devices , and we still have to figure out which of those between `From` and `Dest` are faulty, returns the minimal set of possible faulty devices.

The initial condition of the algorithm is that if the NSGS cannot reach the device x, it calls `generic(NSGS,`x`,`$\emptyset$`)`,

Algorithm 1 The fault diagnostic algorithm

generic(From,Dest,H)
    if (From = Dest) **then** return(H)
    p = path(From, Dest)
    e = select(From,Dest,p)
    if e cannot be queried **then**
        e = substitute(e)
    if (e = Dest) or (e = From) **then**
        return(H)
    else
        q = path(From,e)
        if e is reachable **then**
            $H = H - q$
            generic(e,Dest,H)
        else
            $H = H \cup q$
            generic(From,e,H)

which will automatically set the initial hypothesis H as the path between the manager (device running NSGS) and the faulty element x.

The function `path(From,Dest)` computes the sequence of devices in the path from `From` to `Dest` and back. It deals with cases in which the path of the packet from `From` to `Dest` is not the same as from `Dest` to `From`. If the network model is incomplete in the sense described above it represents that the packet traveled through a collision domain without specifying the details.

The function `select` is a binary search algorithm that selects one element in the path.

Finally, the `substitute` function adds some "thinking ahead" in relation to the choice of the element e. For example if e is a non managed hub, there is no direct way of querying it. Thus `substitute` will return some device which can answers to queries and whose path passes through e. `Substitute` also takes into consideration that there may be computers in the network which are "turn-off-able", that is, personal computers that may be turned off by their users. So there is no point in querying such computers.

Once all queries in the path are made, the GDS returns one minimal set of faulty devices and sends one trouble ticket as output. In the tests we have traced, the most part of the solutions was reduced to one device and its respective cable, which is very precise. However, if the network model is incomplete, the solution loses precision, and the minimal set may contain many devices and their interconnecting cables.

Since the hole on-line system is cyclic and the period is short, one mechanism of control of trouble tickets was developed to ensure that one re-incident fault will not be considered as a new trouble, only advising that the problem has not yet been solved. Moreover, it flags when the faulty device returns to the normal state.

VI. Performance Diagnostic

In the performance diagnosis we assume that performance degradation is usually due to overloading in an collision domain or high equipment CPU usage rate, like a computer with high CPU occupancy rate or a router with high processing rate due to packet filters. We believe that these are the most common causes for performance problems in a stable network, where stable means that the network operates usually without performance problems. The performance degradation may be observed by some symptoms like increase in the communication latency, in packet retransmission, and as an load rate increase within the collision domain.

The assumptions of the performance diagnostic algorithm are:

- TCP/IP network without dynamic routing
- single performance fault during the diagnostic cycle
- for each collision domain it is given a "normal network latency"
- for each collision domain it is given a threshold latency, above which one considers the domain to be experiencing a performance fault
- performance faults are caused by overloaded router or computers sending or receiving too much packets.

Again, we believe the first two assumptions to be reasonable for the reasons already explained. As for the normal and threshold network latency, we believe that these values are a function of the technology used in that domain of collision and the number of devices connected in that domain. We still have to verify if these assumptions are reasonable empirically. Finally, as for the last assumption, corresponds to our experience of performance faults in a stable network.

The performance algorithm is similar to the fault algorithm explained above with the following differences:

- the function `path` does not compute the sequence of devices in the forward and return path. It computes the sequences of collisions domains, and their separators.
- we fixed a forward method in exploring the path.
- the NSGS takes into consideration the latency value of the query to the device, invoking the diagnostic system as soon as it find some unacceptable value.

In the performance algorithm:

- `select-forward` returns the first element of the path
- `member` returns an element of a collision domain which can be queried.
- `latency`(e) is the total latency of a query to the element e.
- `threshold`(d) is the threshold latency for the collision domain d.

VII. Tests

The tests of this system were executed at the LSI-USP LAN (Laboratório de Sistemas Integráveis of the University of São Paulo). This LAN is composed of hundreds of components including three routers, three switches, approximately 12 hubs and various terminals. Some of this

Algorithm 2 Performance Algorithm

```
performance(p, Overhead)
    e = select-forward(p)
    if e is a switch then
        performance(p − e, Overhead)
        else if e is a collision domain then
            e′ = member(e)
            lat = latency(e′)
            if lat − Overhead > threshold(e) then
                return(e)
        else if e is a router then
            lat = latency(e)
            if lat − Overhead > threshold(e) then
                return(e)
            performance(p − e, lat).
```

components are not SNMP-managed. The LAN does not use dynamic routing.

The interfaces to the network, specially the NSGS, were implemented in Java. The modules that construct the model and perform the diagnostic reasoning (the NDS, CCS and the NSGS) were implemented in Prolog. The system runs in a Linux workstation. The NDS, the most time expensive module takes approximately six hours to collect all available information, but the rate in which the NDS queries the various MIB is low so it will not overload the system.

In the LSI LAN it was possible to automatically collect and represent approximately 65 % of the LAN with the NDS algorithm; the remaining information which was mainly connecting cables linking hubs to other hubs and to computers was manually added.

The generation of the poll sequence found 52 valid devices to be polled, so that the entire network could be managed. A large set of hosts (about 30 PCs) were not part of the polling set, since one can not know if they were faulty or simply powered down. In this case, the algorithm ensures that at least one device represents the collision domain. The on-line mode was set to infinitely iterate the poll sequence. The adoption of this polling heuristic decreased the polling period from around 10 minutes when using a commercial management platform to 1 minute.

We have artificially inserted communication errors into this network, by disconnecting cables. The system was able to precisely identify the exact place in which the error occurred, returning the cable or the immediate device connected to it as the source of the error and consequently the result of the diagnostic. The time to diagnose one fault is dependent on the delay of the answers for the ping request, but within one minute of the introduction of the error, the system would produce the trouble ticket.

VIII. Conclusion

This work has merged a theoretical approach from AI with a practical application over a real LAN. The application of the Model Based Reasoning in the diagnosis of computer networks has shown that this approach can be extended to the solution of dynamic systems such as LANs where the model has to be frequently updated. It also provides one feasible way to deal with the constant changes in computer networks, since none of the core algorithms have to be changed if the model changes.

This dependence on the model enhances the importance of a good discovery algorithm. This is a particular subject. The modeling of the network based on the SNMP protocol is a hard task. It involves a great amount of information that is usually bad configured or missing. To maintain the MIBs updated is the unique way in which a manager shall gather consistent and complete information from the network. [9] develop one detailed algorithm to automatically model SNMP based networks. In the application of the discovery system developed here the information regarding connections between devices for diagnosing communication faults is vital. However, this connections are difficult to gather and have to be manually inserted.

We hope to have shown that a model-based reasoning is a promising methodological approach to solving management and diagnostic problems in computer networks. Future work will extend the approach to WAN with dynamic routing, and the experimental evaluation of the performance presuppositions that were assumed in the algorithm. We are also considering adding further information to the model and reasoning about them. This work is the continuation of the work reported in [11].

References

[1] L. Lewis, "A case-based reasoning approach to the resolution of faults in communication networks.," in *ISINM - International Symposium on Integrated Network Management*. 1993, Elsevier.

[2] G. Dreo and R. Valta, "Using master tickets as a storage for problem solving expetise," in *IFIP/IEEE International Symp. on Integrated Network Management IV*, 1995, pp. 328–340.

[3] R. Cronk, P Callahan, and L. Bernstein, "Rule based expert systems for network managements and operations," *IEEE Network*, vol. 2, pp. 7–21, 1988.

[4] R. Goldman and H. Latin, "Automated knowledge acquisition from network management databases," in *IFIP International Symp. on Integrated Network Management IV*, 1991, pp. 541–549.

[5] K. Houck, S. Calo, and A. Finkel, "Towards a practical alarm correlation system," in *IFIP/IEEE International Symp. on Integrated Network Management IV*, 1995, pp. 226–237.

[6] P. Wu, R. Bhatnagar, L. Epshtein, et al., "Alarm correlation engine (ace)," in *Proceedings of the 1998 IEEE Network Operations and Management Symposium (NOMS'98)*. 1998, pp. 733 – 742, IEEE.

[7] M. Klemettinen, "A knowledge discovery methodology for telecommunication network alarm databases," 1999.

[8] H. Wietgrefe and K. Tochs, "Using neural networks for alarm correlation in cellular phone networks," 1997.

[9] H. Lin, S. Lai, P. Chen, and H. L. Lai, "Automatic topology discovery of ip networks," *IEICE TRANS. INF AND SYST*, vol. E83-D, no. 1, 2000.

[10] K. Yoshihara, K. Sugiyama, H. Horiuchi, and S. Obana, "Dynamic polling algorithm based on network management information values," *IEICE TRANS. COMUNN.*, vol. E82-B, no. 6, pp. 868–877, 1999.

[11] J. Wainer, L. Barros, V. Bernal, and M. Lemos, "Network fault diagnosis: A model based approach," in *Proceedings of the 2000 IFIP/IEEE Network Operations Management Symposium*, April 2000.