OPTMIZED DELIVERY MECHANISMS
FOR QOS OF TRAPS

A THESIS

submitted by

S. SREE HARI NAGARAJAN

for the award of the degree

of

MASTER OF SCIENCE

(By Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

DECEMBER 2007

Dedicated

to

Almighty

THESIS CERTIFICATE

This is to certify that the thesis entitled **OPTIMISED DELIVERY MECHANISMS FOR QoS of TRAPS** submitted by **S.Sree Hari Nagarajan** to Indian Institute of Technology, Madras, for the award of the Degree of Master of Science (by research) is a record of bonafide research work carried out by him under my supervision and guidance. The contents of this thesis have not been submitted to any other university or institute for the award of any degree or diploma.

Place: Chennai 600 036 Dr. Timothy A Gonsalves

Date: 14.12.2007

ACKNOWLEDGMENTS

I express my sincere gratitude to my guide, **Dr. Timothy A. Gonsalves**, for his invaluable guidance. I am greatly indebted to him for the time he spent on my thesis amidst his busy schedules. I thank the **faculty and the staff of Computer Science and Engineering** for all the help and facilities extended to me. I thank **Dr. Hema A. Murthy** and **Dr. R. Manivasakan**, for their valuable suggestions during the GTC meetings.

I express my sincere thanks to **Mr. Sanjay Gupta and Midas Communication Technologies Pvt. Ltd., Chennai** for providing me an opportunity to work in the corDECT system. I am thankful my Internal Guide **Mr. A. Balaji** for supporting me to complete my thesis and especially for his guidance in presenting this work. My special thanks to **Mr. C. Jagadish** for helping me in modeling the sensitivity analysis measurements in the corDECT system and helping me in presenting this work in its present form. I am grateful to **Mr. Rene Abraham,** CTO Midas Communication Technologies Pvt. Ltd., Chennai, for his support, **Mr. M. Murali**, Head-Admin, Midas Communication Technologies Pvt. Ltd., for providing me with the resources required during the implementation phases. I thank my friend **Mr. Vinod Kumar Jammula, colleagues** and **all well wishers** for encouraging me to complete my thesis.

Last but not the least, my special thanks to my **parents, sister, grandmother** and all **family members** for their support and prayers. I also thank them for their wonderful attitude of accepting my negligence towards them without any resentment.

ABSTRACT

With the growing reliance on telecommunication, it is of paramount importance that the telecommunication network operators maintain high availability and low mean time to repair (MTTR). To achieve this objective at a low operational cost, central network management becomes mandatory. The elements of the network use traps to convey any problem to the central management station. Often there is a possibility that a failing network element (NE) floods a management station with redundant traps and congests all available network bandwidth. When there is a flood of traps, the manager gets overloaded and critical traps might get lost or delayed. The analysis of trap data from multiple corDECT telecom networks confirm that a major portion of traps during a flood is contributed by a few repetitive events from a small number of NEs. Operator should be presented only with the actual faults without repetition. Methods are needed to ensure reliable and timely delivery of traps in this distributed network.

We propose a correlation engine that correlates and removes the repetitive and cleared traps that are received at the manager so as to present only the actual faults to the operator. We propose a technique by which the manager can control the NEs that are flooding it with traps. Apart from this, to avoid forwarding of short lived and redundant traps we propose distributed correlation and compression techniques at the NEs. These techniques put together helps reduce the total number of traps delivered and give control to the management station to maintain QoS for trap delivery. This also ensures that the management station is not overloaded.

The proposed rule-based correlation is implemented and is being used in all corDECT deployments. It is seen that the correlation engine is very effective in reducing the number of traps presented to the operator. The other proposed techniques are implemented in a corDECT network both at the NMS manager and device side. Using experiments in a laboratory setup various parameters for the scheduling algorithm are tuned to obtain optimal results. The field trace driven experiments show that there is considerable gain, in terms of control at the manager, number of events reported to manager and timely trap delivery, even under heavy flow of traps.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

NE	Network Element
DECT	Digitally Enhanced Cordless Telephony
DIU	DECT Interface Unit
LAN	Local Area Network
TDM	Trap Delivery Mode

NOTATIONS

N_{NE}	Number of NEs
IPI	Inter-Pull Interval
LCE	Local Correlation Engine
LCEHT	Local Correlation Engine Holding Time
AR_{HLT}	Trap Arrival Rate of Healthy NEs
AR_{FLT}	Trap Arrival Rate of Faulty NEs
TH_{TDM}	Threshold for Changing Trap Delivery Mode

CHAPTER 1

INTRODUCTION

Telecommunication growth in developing countries, like India, is fuelling the economy to a new high [1]. With a host of services being provided like e-agriculture, e-healthcare and e-governance the reliance on these systems (both hardware and software) is becoming very high. To ensure high availability of telecom networks it becomes evident that these systems must be monitored and managed effectively and efficiently. To reduce operational costs, these systems are often managed from a central location.

1.1 The corDECT Wireless in Local Loop System

With the cost of wireless technology systems decreasing steadily, it would be economical to adopt wireless technology especially in sparsely populated rural areas. To tap this potential of wireless technology and to increase the tele-density in the rural areas at an affordable cost, the TeNeT Group of IIT Madras and Midas Communication Technologies (P) Ltd, Chennai have developed the corDECT Wireless in Local Loop system [2]. It acts as an Access Network (AN) connected to a Local Exchange (LE). corDECT is based on the DECT standard [3]. It is a cost-effective WLL system that is widely deployed across 25 countries including India to serve the data and voice needs of urban and rural population. This DECT based system provides simultaneous toll quality voice and broadband Internet access to the subscribers.

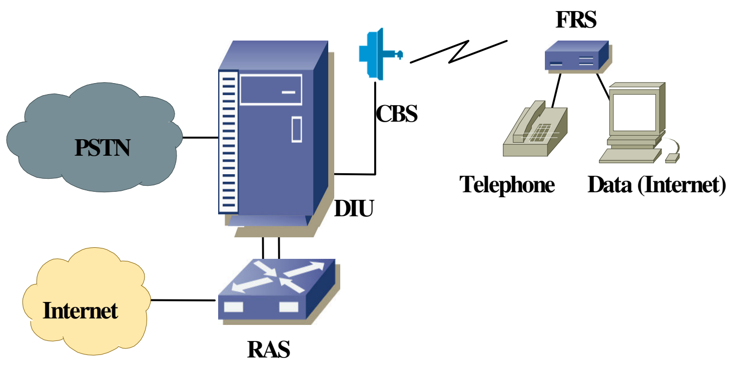


Figure 1-1 corDECT system architecture

A single corDECT system can provide up to 5000 subscribers connections. The major subsystems of corDECT are shown in Fig 1-1.

DECT Interface Unit (DIU): It performs system control and interfaces to parent PSTN network.

Compact Base Station (CBS): It can provide wireless access to subscribers in the radius of 10 kms and are in line-of-sight. A single CBS supports up to twelve simultaneous connections

Fixed Remote Station (FRS): It is a fixed wireless terminal that relays the voice and data traffic between the subscriber end and the DIU

Remote Access Switch (RAS): It carries the IP data to the Internet backbone

Base Station Distributor (BSD): It is designed to extend corDECT coverage to pockets of subscribers located far away from the DIU. On one side it connects to the DIU over E1 lines and on the other side it supports up to four CBS.

Relay Base Station (RBS): It is a wireless relay unit used to extend the wireless coverage of a corDECT system up to 35 kms.

1.2 Network Management System (NMS)

A Network Management System (NMS) is a combination of hardware and software used to monitor and administer a network. An effective network management system can result in a significant improvement in productivity through better utilization of the available resources. The goal of network management is to ensure that the users of a network receive the services with the quality of service they expect [4]. It follows that improvement in the quality of services offered to the users will result in more usage and hence more revenue. Savings due to lower maintenance and operational costs can also be reaped. The functions of NMS are categorized into five heads referred to as FCAPS. They are

- Fault Management.
- Configuration Management
- Accounting Management
- Performance Management
- Security Management

Communication between a manager and a NE happens in two ways. When the management station requires some data from a NE it sends a request to the NE. The NE responds to the request by providing the information requested for. Traps [5] are used by the NE to convey asynchronous events to the management stations. For example, when a fault occurs at the NE it generates a trap to the management station.

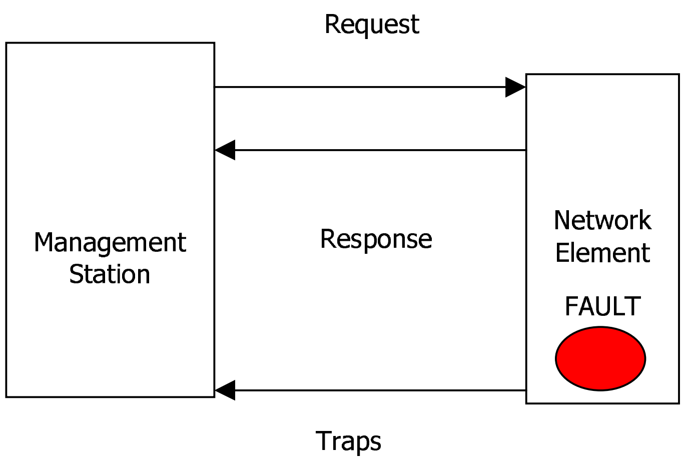


Figure 1-2 Generic Network Management Model

This is shown in Figure 1-2. It is evident that traps are an important cog in the management wheel.

SNMP [6] is the widely deployed network management protocol defined by IETF. Almost all data and telecom networks support SNMP-based management. Different request messages supported by SNMP [6] include get, get-next, set and get-bulk. It supports traps for events communication from NE to NMS. SNMP [6] uses UDP as the transport protocol. Due to this delivery of traps or request-responses is not guaranteed.

1.3 Problem Definition

One of the most important areas in network management is fault management. Faults are to be delivered in a reliable and time-bound manner to the management station. Often there is a possibility of flood of traps which could saturate the network bandwidth and the processing capability of the management station. Also, because of using UDP as transport protocol traps can be lost during these conditions.

Trap delivery mechanisms are needed that can control and avoid flood of traps. These trap delivery mechanisms should give a fair chance to all NEs in the managed network. It should be possible to ensure that the trap delivery happens in a time-bound manner. Also *QoS* measures like defining an upper bound on time for delivery of traps should be possible. Achieving these objectives for a distributed network with a large number of NEs is a challenging task.

1.4 Contributions

This thesis addresses the issues concerned with optimized trap delivery especially in the face of a flood of traps. Suitable trap delivery mechanisms and techniques to ensure optimized and timely delivery of traps are developed. The proposal is implemented and its effectiveness is verified.

The major contributions of this thesis include the following

- Study of existing trap delivery alternatives
- Analysis of fault data from various corDECT telecom networks
- Design and development of optimal mechanisms for
    - Correlating and removing repetitive and cleared traps so as to present only the actual problem list to the operator
    - Management Station controlling the rate at which the NEs sends traps
    - Fairness to all NEs for trap delivery
    - Reducing traps forwarded at NEs by distributed correlation and compression

    - Providing QoS for traps delivery by adhering to an upper-bound on time for delivery of traps
- Fixing optimal values for parameters of the proposed mechanisms using data obtained from field
- Implementation of the proposed techniques in corDECT system
- Evaluation of the developed system with field traces driven experiments.

1.5 Thesis Organization

Chapter 2 presents an overview of the existing mechanisms for trap delivery. A brief description of the corDECT Network Management system is also presented in this chapter. Chapter 3 presents the findings in the survey of field data of corDECT installations. The details of the proposed mechanisms for optimized trap delivery are discussed in Chapter 4. Chapter 5 describes the design and implementation details of the proposed mechanisms. Chapter 6 describes the simulations, experiments done and the results obtained using the proposed mechanisms. Finally, Chapter 7 presents a summary and conclusions of the thesis with a brief discussion of possible further research.

CHAPTER 2

BACKGROUND

A good understanding of the existing trap delivery mechanisms used in Network Management Systems will help us in setting the required background for the thesis. This chapter provides a comparative overview of various trap delivery mechanisms. As the proposed work is employed in corDECT network, a functional description of the corDECT network and corDECT network management architecture are presented. As throughout this thesis we will deal with traps, the various fields present in a typical corDECT trap are given at the end of this chapter.

2.1 Existing trap delivery and control mechanisms

This section explains about various trap delivery mechanisms that are used for Network Management. Finally it presents a comparison of the same and highlights the problems in the available mechanisms.

2.1.1 SNMP traps over UDP

UDP [8] is the standard trap delivery mechanism used by popular SNMP [6] implementations like NET-SNMP [7]. The NE sends a trap to the manager IP address (typically on port 162) with UDP [8] as the underlying protocol. This is shown in Figure 2-1.

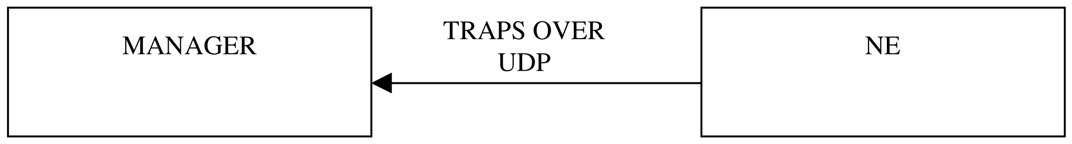


Figure 2-1 SNMP traps over UDP

2.1.2 SNMP traps over TCP

TCP [9] mechanism is costlier but more reliable than UDP [8]. NE sends a trap to the manager IP address (typically on port 162) with TCP [9] as the underlying protocol. This is shown in Figure 2-2.

2.1.3 Store And Forward Traps (SAFT) based Trap Delivery

Store And Forward Traps is a mechanism used for delivering traps over an intermittent Data Communication Network (DCN) [11] [12]. An intermittent DCN [12] is one where the connection between manager and NE is established on need basis. The connection is initiated by the manager when it wants to retrieve some data from the NE. Similarly the connection is initiated by the NE when it has traps to deliver to the manager. Due to the intermittent nature of DCN [12] as there could be losses in traps over such DCNs [12], the traps are stored in an ASCII file and transferred to the management station once the connection is established. At the manager end, a trap regenerator reads the traps from the ASCII file and regenerates them towards the manager. Hence the underlying DCN [12] and trap delivery mechanism are transparent to the manager. Though this mechanism was initially conceived for intermittent DCNs, [12] it is being used in all deployments of corDECT systems, to have a uniform and reliable trap delivery mechanism for all DCN types.

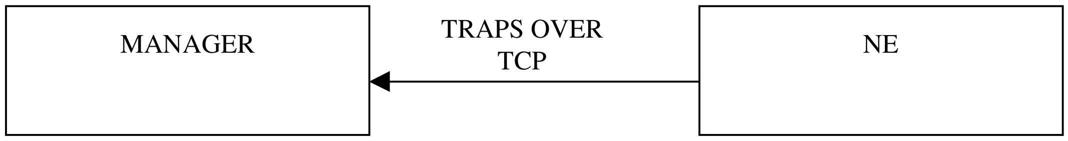


Figure 2-2 SNMP traps over TCP

In case of a dial-up DCN [12] due to inherent limitation of number of telephone lines at the manager, usually one or two devices can only transfer traps simultaneously. However in case of a LAN, the manager is simultaneously connected to a large number of devices. In this case, the number of NEs that could push traps simultaneously is high. This can lead to a high load at the manager. This is controlled by placing a limit on the number of connections accepted at the management station for traps delivery. The basic architecture of trap delivery using SAFT [11] is shown in Figure 2-3.

2.1.4 RFC 1224

RFC 1224 [10] deals with control of trap delivery from managed network. Realizing the need for controlling the flow of traps from NEs to protect the manager, RFC – 1224

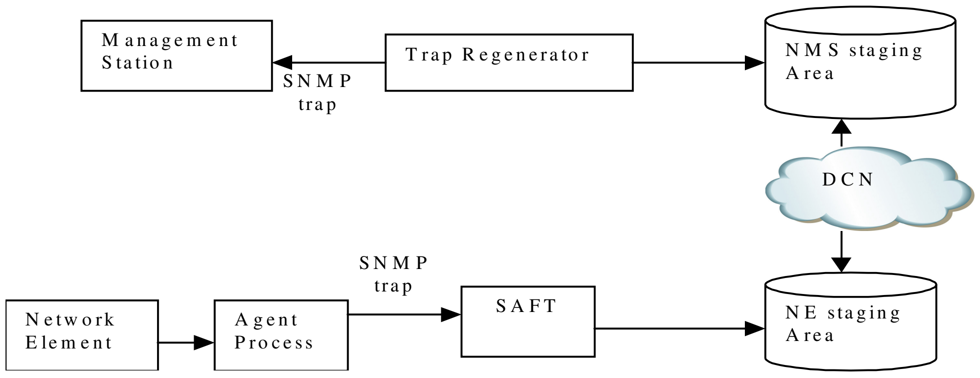


Figure 2-3 SAFT based trap delivery

advocates a two pronged strategy [10]. The first strategy is to stop sending traps from a NE if the number of traps within a time-window is more than a threshold set by the manager. The time-window and the threshold are both manageable attributes. When this threshold is reached, the NE sends an alerts disabled trap and stops forwarding traps until the manager sets a flag indicating to the NE to forward traps again.

To thwart loss of data, as a second strategy it advocates storing of all trap data that are not forwarded and make them retrievable over SNMP [6] so that the manager can get all the missed traps at its convenience.

2.1.5 Shortcomings in Trap Delivery Techniques

The UDP [8] mechanism though inherently simple is unreliable. TCP [9] mechanism is reliable but costly and not widely used. Its use is suggested only for traps that are absolutely vital for the proper functioning of the management system. (For e.g., alerts disabled trap of RFC-1224 [10] is suggested to be sent over TCP [9] wherever available).

RFC-1224 [10] saves the management station from a flood of traps and provides a mechanism to retrieve all the traps. But the number of traps is not reduced and it does not provide a scope for limiting trap generation.

The SAFT [11] based approach overcomes the problem of using the unreliable UDP [8] by storing the traps in files and transferring them. But it leaves open the possibility of a misbehaving-NE causing heavy load at the manager when it is used in an environment with a permanent DCN [12]. Due to the limitation on the number of simultaneous connections accepted at the NMS there is a possibility that some NEs may be starved for the channel to deliver traps.

Out of the mechanisms discussed above, the SAFT mechanism is versatile for use in different DCNs and reliable. If the limitations of SAFT are countered, it will be a very effective trap delivery mechanism. Through the rest of this thesis we will provide mechanisms to overcome the limitations identified above.

2.2 The corDECT Network Management System

The corDECT NMS provides comprehensive operation and maintenance functionality through a menu-driven Graphical User Interface (GUI). The objective of the management system is to aid in providing improved Quality of Service (QoS) at a reduced operational cost. Its repertoire includes full-fledged fault, configuration, accounting, performance, security operations, event logging facility, versatile DCN [12] connectivity options and extensive report generation mechanisms.

The corDECT NMS is capable of managing multiple co-located or distributed corDECT systems. NMS redundancy is an added, optional feature.

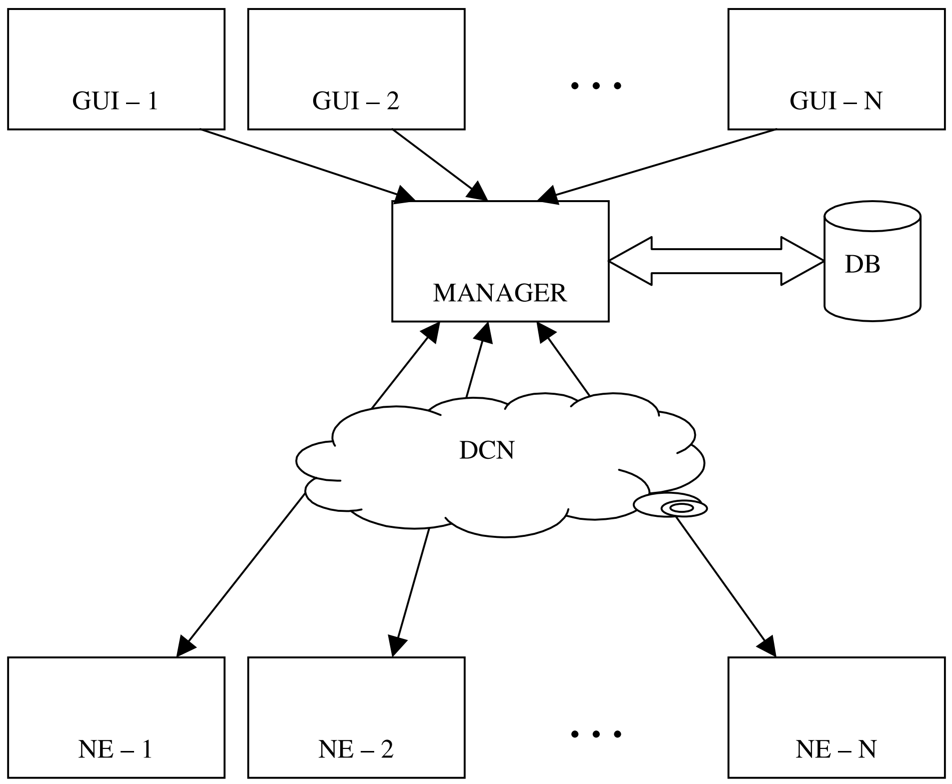


Figure 2-4 corDECT NMS architecture

2.2.1 corDECT NMS Architecture and features

The architecture of corDECT NMS is shown in Figure 2-4.The corDECT NMS is a full-fledged Network Management System used for the remote monitoring and management of the corDECT systems. It is built based on the client server architecture.

The corDECT NMS server supports multiple GUI clients and can manage multiple corDECT systems simultaneously. It has a centralized RDBMS to store all the user data, configuration data and traps generated from the corDECT systems. Centralized AAA (Authorization, Authentication and Audit) functionality is supported at the server. It provides a standard North Bound Interface (NBI) to connect to higher level NMS systems. It is capable of managing the corDECT systems of different versions and of different PSTN connectivity (V5.2, R2MF and SS7) simultaneously.

A single NMS server can have multiple clients, and multiple operators can operate on separate client machines. This gives enormous flexibility to the operator to simultaneously manage the system on various fronts. For example, one client can be used/dedicated for the subscriber provisioning activity, another for the field operations to monitor the performance of the system while a third client can have its own terminal for fault monitoring.

2.2.2 Fault management in corDECT NMS

All the traps generated from the corDECT system are classified according to the TMN guideline [5]. Top level view of the corDECT NMS represents the consolidated status of all the managed the corDECT systems. Zoom in facility is provided to know the status of each subsystem of the corDECT system. It has the facility to view all pending and history traps specific to each subsystem with a click on the subsystem. Status is updated based on polling and traps. Trap states such as alerting, acknowledged, zombie and cleared are maintained. It has the facility to filter and view traps based on different criteria such as severity, NE, subsystem, time and trap state. It has trap counters to represent count of traps of different severities. Different colors are used to represent different severities. On receipt of a trap, it can be configured to do one or more of the trap actions, such as playing-wave-file, mail, SMS and print. It has the facility to filter the traps based on the user criteria. It has a rule based trap escalation policy.

2.3 Traps in the corDECT System

The fields in a typical trap are given below.

Managed Object Class – MOC: This field indicates the class or entity that generated the trap. An example of MOC can be a sub-system card in a telecom network.

Managed Object Instance: This field indicates the instance number of the entity from which the trap has emanated. An example could be card number -1.

Event type: This field indicates the type of event that caused the trap to be generated. Examples of event type include equipment alarm, communications alarm, environmental alarm etc.,

Probable cause: The probable cause further qualifies the event type. An example of probable cause for event type equipment alarm could be equipment failure.

Specific Problem: The specific problem pinpoints the reason for the failure and may be technology specific. An example of specific problem for probable cause equipment failure could be Power failure.

Perceived Severity: This field indicates the severity of the situation arising out the event that is being reported. It can take values of critical, major, minor, warning, cleared and indeterminate. Critical severity is used when the fault has resulted in a service-affecting condition and warning is used when there is a scope for development of a future service affecting condition for which this event could be a forerunner.

Monitored Attributes List: This field carries a set of attributes that are of interest when an event has occurred. For example, when a sub-system boots up the version number of the software running in the sub-system could be sent in the trap as a monitored attribute.

2.4 Summary

This chapter has introduced various trap delivery alternatives used for Network Management. We compare and contrast the alternatives and present the limitations. A brief summary on the architecture and features of the corDECT Network Management System and the different fields in a typical corDECT trap are also presented.

CHAPTER 3

FIELD DATA ANALYSIS

We have taken trap data from various corDECT deployments for analysis. Data pertaining to traps from these installations will help us understand the ground realities and help us develop a sound solution for the problems highlighted in Section 2.1.5

3.1 Evolution of Fault Management in corDECT NMS

Earlier the corDECT NMS used to present the traps to the operator as they were delivered at the management station. The trap indicating the failure and the trap indicating the corresponding clear were both presented to the operator. With the increase in size of the network, the trap arrival rate increased and the trap list presented to the operator grew bigger. It was very difficult for the operator to comprehend the list that was presented and to identify the real problem indicating traps from the non-faulty traps.

We studied the corDECT system and the events that are generated and found that a rule-based correlation engine to correlate the events in the corDECT network could be useful in managing it. Based on this we have developed a rule-based trap correlation engine. With the use of this correlation engine only the fault indicating traps are shown to the user. When a trap that signaled the end of the fault indicating condition is received, the fault indicating trap is removed from the operator's view. The challenges we faced in developing a rule-based correlation engine are

- ,Defining a comprehensive rule set for correlation of all kinds of corDECT traps taking into account the architecture of corDECT system and subsystems
- Considering the probability of various traps, designing an optimal algorithm for correlation
- Optimal usage of RDBMS features for optimum performance and ease of implementation
- The DIU has 2 PCs (one in active mode and other in passive or stand-by mode) for redundancy to thwart a single point failure of the system. So both the active and the passive copies will report an event. Taking care to correlate redundant/repeat traps from both the active and passive copies.

The architecture of correlation is shown in Figure 3-1. The proposed correlation mechanism was implemented in corDECT NMS and the same is being used in all corDECT deployments. With the introduction of correlation in corDECT NMS, there was a substantial improvement in the number of traps that was presented to the user. Table 3-1 conveys the same statistically.

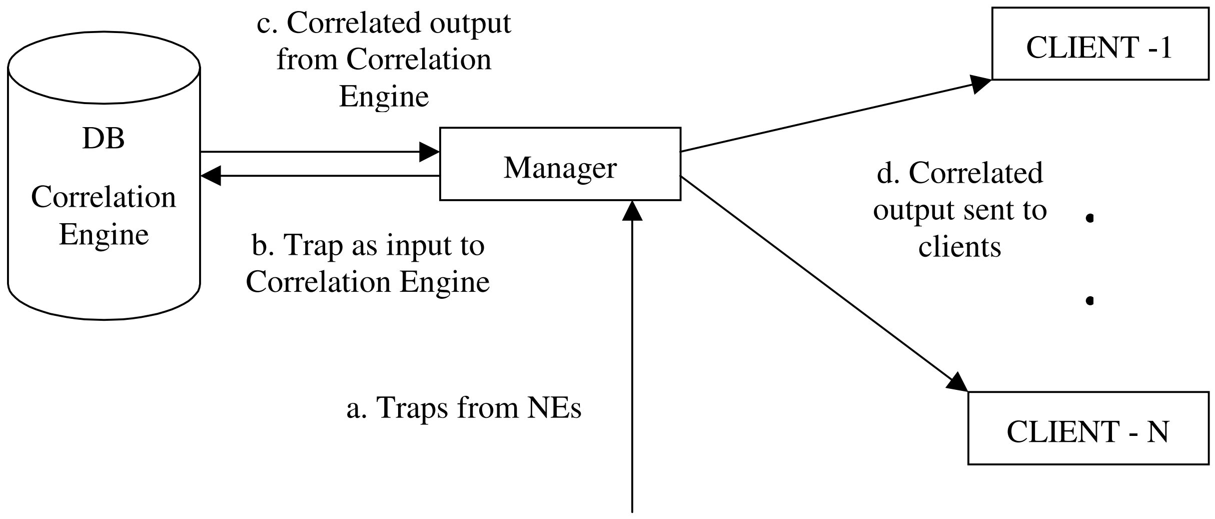

c. Correlated output from Correlation Engine
DB
Correlation Engine
Manager
b. Trap as input to Correlation Engine
a. Traps from NEs
CLIENT -1
d. Correlated output sent to clients
CLIENT - N

Figure 3-1 Correlation Architecture in corDECT NMS

Table 3-1 Comparison of traps presented to user before and after correlation

EMS No.	**No. of NEs in the network.**	**Number of traps presented to operator before correlation**	**Number of traps presented to operator after correlation**
1	54	< =150,000	200-300
2	100	<= 200,000	400-500

Table 3-2 Details of NMS sites from which trap data was collected for 4 weeks

EMS No.	**No. of NEs in the network.**	**Location**
1	11	Dhaka, Bangladesh
2	61	Mohali, India
3	100	Jaipur, India

3.2 Current situation

To find out the current status with respect to traps, data from three corDECT NMS sites was collected for 4 weeks and analyzed. The details of the three corDECT NMS sites are given in Table 3-2. Each of these NEs serves around one-thousand subscribers. Data from the sites shows that because of correlation less than 1% of total traps received are presented to the operator as faults. Though the number of traps presented to the user is low, the actual number of traps being transported from the NEs to the NMS is high. The following sections give the statistics, the trap arrival rates at NMS and results of other analysis done on the data.

3.2.1 Trap Arrival Rates

From the collected trap data, various statistics are computed. The statistics are tabulated in Table 3-3.

Table 3-3 Traps Arrival rate from different sites

	Total DIUS		
	11	**61**	**100**
Total Traps in network	286109.0	612246.0	1185746.0
Network Average/Day	9536.9	20408.2	39524.8
Mean/DIU/30 Days	26009.9	10036.81	11857.4
Mean/DIU/30 mins.	22.2	8.3	8.8

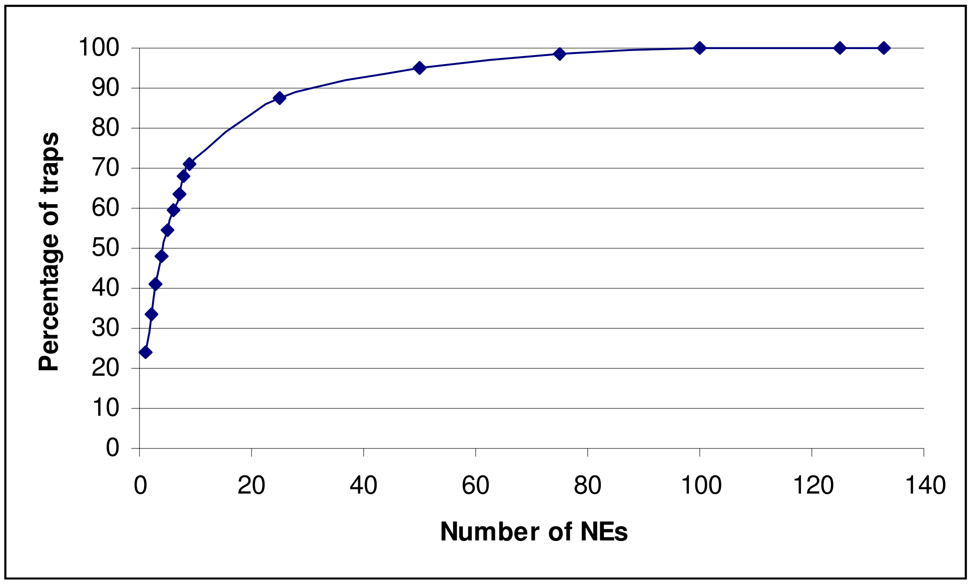


Figure 3-2 Distribution of NEs contributing to total traps in a corDECT network

From the analysis of data from different networks, we see that a large portion of the traps is contributed by only small number of DIUs. As shown in Figure 3-2, we see that more than 60% of traps are generated by only 10% of NEs. This is consistent across different corDECT telecom networks. These 10% of NEs generate traps at an average rate of 51 traps every thirty minutes. Further we see that remaining 90% of the NEs contribute to only 40% of the traps, with an average of approximately four (3.73) traps every thirty minutes. Based on this we categorize NEs into two types.

a) Healthy NEs generating less than four traps every thirty minutes.

b) Faulty NEs generating more than fifty traps every thirty minutes.

We denote the arrival rate from healthy NEs as AR_{HLT} and arrival rate from faulty NEs as AR_{FLT}

3.2.2 Contribution of different trap categories

To understand the contribution from different trap categories we have taken trap based statistics for each network. We categorize traps into three types. Traps that notify of failures are classified as "Failure" traps and the traps that signify any fault-resolved condition are classified as "Cleared" traps. The traps that are repeated are classified as "Repeat" traps. Figure 3-2 shows the contribution from each category of traps as seen in

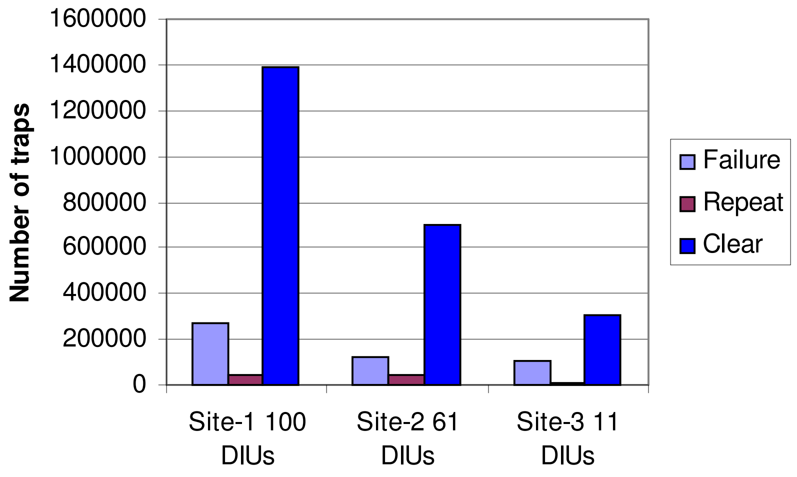


Figure 3-3 Correlation Types of traps for 3 NMS sites

all the three networks. Interestingly it is seen that the distributions of various trap types were found to be similar across different networks.

Analyzing the data on a whole from three sites we see that 4% of traps are repeat traps, 17% of traps are failure and 79% of traps are clear traps. Another interesting observation is that all the failure traps were correlated by clear traps generated from the same NEs and not because of any network-topology based information available at the management station.

3.2.3 Life time of Failure Traps

An analysis was done to find out after a failure event was reported, how long it took for the faulty condition to be rectified – based on the time difference between time of occurrence of an event and the time of occurrence of a subsequent event that eventually cleared the faulty condition. The statistics of this were again found to follow the same pattern across the three sites and the consolidated view is shown in Figure 3-3. The results show that the overwhelming majority of traps are only indicating faults of a transient nature and the fault condition gets restored to normality quickly. For example 88% of failures are resolved within ten seconds and 94% gets resolved within twenty-five seconds.

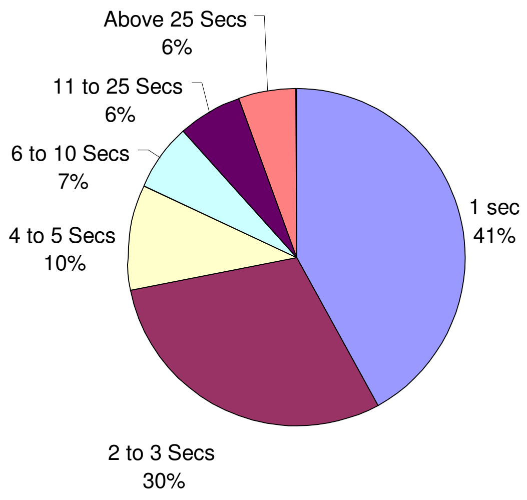


Figure 3-4 Lifetime of failure traps for 3 NMS sites

3.3 Summary

This chapter has traced the evolution of fault management in corDECT upto its present form. Data collected from 3 different telecom networks sites for 4 weeks was presented and analyzed. A summary of the collected data and the analysis done on them were presented in this chapter.

CHAPTER 4

PROPOSED MECHANISMS

We presented the survey of field data of different corDECT installations from a trap-handling standpoint in the previous chapter. With this background, we propose mechanisms for optimized trap delivery.

4.1 Introduction

We have discussed the problems in existing trap delivery mechanisms in Chapter 2. As discussed we have identified SAFT as the suitable trap delivery mechanism both in the case of permanent DCN [12] as well as intermittent DCN [12] such as dial-up. Here we summarize the limitations with the existing *SAFT-based trap delivery* mechanism.

- Lack of control at manager – Manager has no control over the number of traps delivered by a NE. Due to this a faulty NE can flood the network.
- Unfairness – None of the trap delivery mechanisms offer a scope for fairness for trap delivery from different NEs. It is distinctly possible for the traps of an NE to be delayed for a long time.
- Limiting number of traps forwarded from NE – The field-data survey indicates that there are a huge number of redundant and short-lived traps. This gives a scope for limiting the traps forwarded from the NE by employing some crunching mechanism at the NE.

- QoS of traps – In the context of this thesis we define QoS of traps as the time-bound on delivery of traps. In the existing trap delivery mechanisms no such upper bound on trap delivery time can be ensured.

In the rest of this chapter we propose techniques to overcome the said limitations with the *SAFT-based trap delivery* mechanism.

4.2 Controlling Traps Arrival Rate at the Manager

Normally, when an event that is of importance to the manager occurs at an NE, the NE forwards it as a trap to the manager. When there is a flood of events, they result in a flood of traps. The problems become more acute when multiple NEs have a flood of events simultaneously. When this happens there is a distinct possibility that the manager is overwhelmed by the sudden spate of incoming traps. To counter this, we have defined a MIB attribute named "*trapForwardType*" which is maintained at the NE. This parameter can take two values viz., Push and Pull.

4.2.1 Logic at NE

When the value of *trapForwardType* is set to Push, the NE sends the traps to the manager as in SAFT [11] based trap delivery mode. On the other hand if it is set to Pull, the NE writes the traps in a file and a registration trap is sent periodically to the manager indicating that there are traps to be pulled from that NE. Now it is up to the manager to pull the traps from the NE. Sending of this periodic registration trap continues until the trap file is pulled by the manager.

4.2.2 Logic at Manager

The manager maintains a counter of traps from each NE. When this counter's value goes above a threshold for a given NE, the manager sets the trap forward type of that NE to "Pull". Once this is done, the manager periodically *pulls* traps from such NEs whose trap delivery mode was set to "Pull". Now, as the manager controls the rate of traps from the managed network, a misbehaving NE cannot overwhelm a manager. The threshold, which is used to decide the trap forward type of a NE, is termed as *Th*reshold for *T*rap *D*elivery *M*ode *(TH_{TDM})*. The periodicity of pulling traps is termed as *I*nter *P*ull *I*nterval (*IPI)*

4.3 Fairness in Scheduling Trap Delivery

The manager maintains the list of NEs whose *trapForwardType* is set to Pull. The scheduling algorithm considers multiple parameters to provide fairness to all NEs for trap delivery. The parameters include contents of the registration trap received from NEs, importance of the NE as configured by the operator and number of times a NE has already been scheduled to pull traps.

The registration trap sent by a NE has the following fields.

1. Time of generation of registration trap
2. Cumulative severity of the traps that are present at the NE
3. Traps count for
    a. Critical traps

b. Major traps

c. Minor traps

d. Warning traps

e. No severity (Information events) traps

Some parameters that could determine the importance of a NE include but not limited to the ones listed below.

1. The number of users being serviced by the NE
2. The revenue obtained from the NE
3. Servicing any VIP/VVIP etc.,

Each NE is given a rank as per the equation [4-1].

$$NE_{rank} = f\,(T_{Reg} , S , C_{Sche} , I) \qquad [4.1]$$

where *NErank* is the rank of a given NE, T_{Reg} is the time of generation of registration trap stored as seconds since epoch, S is the severity reported by the NE. Higher the severity lesser the value. C_{Sche} is the number of times the NE is scheduled for Pull from the last change in "trapDeliveryMode" , I is the importance of the NE as configured by the operator. Higher the value of I higher is the importance.

The NEs with good ranks (smaller the number better the rank) are scheduled in a round-robin fashion. After a NE's traps are pulled from, the parameter "number of times scheduled" for the NE is incremented by one so that other NEs that are not yet scheduled will have a better chance of making it schedule list by qualifying as "best" NEs.

After completion of each pulling cycle the scheduling algorithm updates the trap delivery Mode of the NEs (to Push or Pull) based on the number of traps received from the NEs during the last *IPI*.

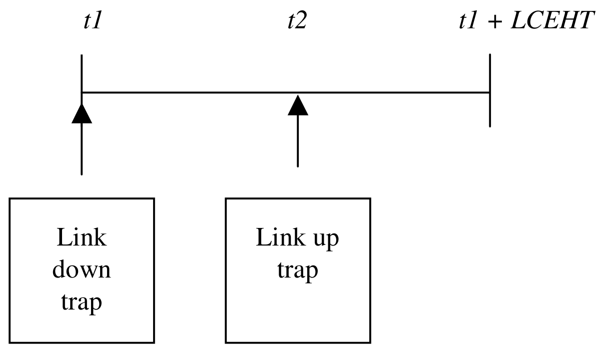
4.4 Limiting the Number of Traps Forwarded from an NE

As was seen in section 3.2.3 a sizeable portion of the Failure traps can be dropped at the NE if we employ some filtering techniques, as the faults that cause these traps are transient in nature. However this needs to be done without affecting the QoS. We propose two methods for limiting the number of traps forwarded from NEs namely correlation and compression.

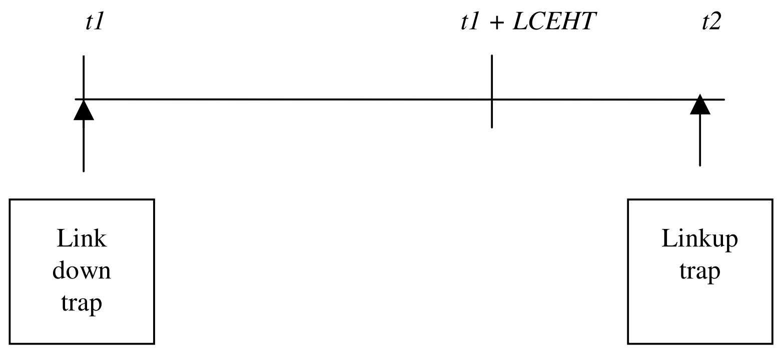
4.4.1 Correlation

Correlation is a process of identifying and removing fault traps on receipt of a trap that signals the end of the faulty conditions. Each trap is given an identity termed the "*trap id"*. For e.g, a trap which is emanated from Subsystem – A with a severity S, of event type E, because of probable cause P is tagged with trap id T. The trap id is associated with a rule-set termed "*correlation rules*" which are of the form 'source trap id" "action" 'target trap id". For e.g., a rule can be 1 clears 2. This means reception of trap with id 1 denotes the end of the fault that trap with id 2 represents.

In the NE before forwarding the traps, they are held in memory for some time which is referred to as Local Correlation Engine Holding Time (*LCEHT*) during which they can be correlated. At the end of *LCEHT* only the traps remaining after correlation are sent to the



a) Correlate and drop traps



. b) Forward both traps

Figure 4-1 Effect of correlation on traps at different times

manager. For e.g., as shown in Figure 4-1, say at time *t1* a link down trap is received and at time *t2* the corresponding linkup trap are received. As shown in Figure 4-1, depending on arrival time of second trap either none of them are sent (Figure- 4-1 a) or both the traps are sent to the manager (Figure- 4-1 b). To ensure consistency and ease of implementation the correlation rule set used at manager is used at the NE.

It must be noted here that the correlation logic can be applied only to those failure traps that can be cleared by another event generated by the same NE. Any higher level correlation like correlation based on network-topology can be done only at the manager.

4.4.2 Compression

As seen often when there is a flood of traps, the same set of traps are repeated (almost) continuously. These traps can be compressed into a single trap before being forwarded to the manager. For e.g. if at times *t1, t2, t3* and *t4*, the same trap is received (which can be identified uniquely by using the trap id), they are compressed and only the first trap generated at *t1* is sent. When this compression is done, the manager will not know that this trap has occurred four times. To reduce this loss of data, the traps are augmented with additional attributes such as repeat count and last occurrence time. In this case the value of repeat count will be three and last occurrence time will be t4.

4.5 QoS for Trap Delivery

The objective of QoS of trap delivery is to ensure a time-bound delivery of traps. With the proposed scheduling algorithm at the manager, for a given *IPI* and a given percentage of faulty NEs, the upper bound on the time for delivery of traps can be defined. As the scheduling algorithm takes into account the time of generation of registration trap and the number of times a NE is scheduled for pulling traps from, it ensures that all the NEs are given a fair chance.

Let us assume that in each *IPI* 10% of NEs traps are pulled. If there are more than 10% of NEs to be pulled from then some NEs would be pulled from in the second and subsequent pull periods. For e.g if there are hundred NEs and twenty have sent registration traps simultaneously, then ten NEs will be pulled from during the first pull period that falls immediately after the receipt of registration traps. The rest of the NEs would be pulled from in the next *IPI* period.

Based on this we can define the upper bound on trap delivery T_{TD} as below.

$$T_{TD} = t + (P_i \text{ X } t) \qquad [4.2]$$

Where T_{TD} is the time for trap delivery, t is the *IPI,* and P_i is the maximum number of *IPIs* after which manager will pull traps from a NE.

4.6 Fixing values for parameters

The various parameters involved in the algorithms/techniques discussed above are

- Threshold for changing trap delivery mode to pull and push (TH_{TDM})
- No. of NEs to pull traps from in one schedule (N_{NE})
- Local correlation Engine Holding time (*LCEHT*)
- Inter-Pull Interval (*IPI*)

In this section we will determine the optimal values for these parameters.

4.6.1 Fixing the value for TH_{TDM}

We saw in Section 3.2.1, the trap arrival rate from NEs (DIUs) of different sites. We saw that the average number of traps from a healthy NE i.e., AR_{HLT} was four every half-an-hour. So if the number of traps from a NE is more than four in last half-an-hour, then the trap delivery mode can be changed from 'Push" to 'Pull". When the number of traps from NEs whose trap forward type is 'Pull" is less than four in half-an-hour, its trap forward type could be changed to 'Push". To avoid thrashing between the two modes of trap delivery, the switch to 'Push" mode shall be done only if the number of traps from the NE is less than four in two consecutive half-an-hour periods. For counting the number of traps from a NE, both the number of traps and the value of repeat count attribute added in traps because of compression, are to be considered.

4.6.2 Fixing the value for N_{NE}

We saw in Section 3.2.1, 10% of the managed elements generates more than 60% percent of the traps. We have seen that this statistic is consistent in all deployments. Based on this we define the maximum number of NEs to be pulled from in any interval as 10% of the total managed NEs. A manager that manages hundred NEs, may have to pull from ten NEs every *IPI*. It is appropriate to note that the arrival rate seen from the faulty NEs, AR_{FLT} was more than fifty-one. Based on the TH_{TDM} defined above these NEs will be put into 'Pull" mode of trap delivery.

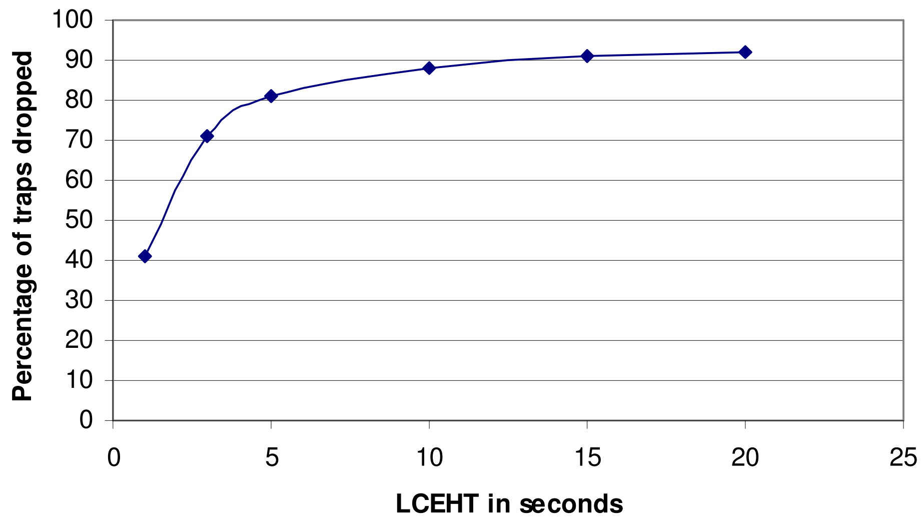


Figure 4-2 Effect of Local Correlation Engine Holding Time on traps dropped at NE

4.6.3 Fixing the value for *LCEHT*

In the analysis of data obtained from various sites, we saw in Section 3.2.3, that 88% of Failure traps can be dropped at NEs if they are delayed by ten seconds and then correlated. Figure 4.1 shows the effect of *LCEHT* in removing traps. There is minimal gain in terms of traps dropped by delaying the traps for more than ten seconds. Based on this we have fixed ten seconds as the *LCEHT* to be used in correlation. More over any event shown for less than ten seconds before being correlated will not be useful for the operator. It must be noted that to obtain best results compression is done just before the traps are pulled by the manager.

4.6.4 Fixing the value for *IPI*

IPI can be decided based on the response time required and the load permissible and capacity of the manager PC. It must be noted that since the *IPI* determines the frequency of pulls to be done from NEs that are already misbehaving by sending traps at a higher rate than is normal, it would be prudent to have a higher value for *IPI* so that the distributed correlation and compression logic can come into effect and reduce the number of traps that are forwarded from the NE. But on the flip side, having a very high value for *IPI* would mean a longer response time for some important traps that may be present among the "noisy" traps that are generated. It must be noted here that the minimum value for *IPI* can be no less than thirty-seconds as it is the time taken to pull traps from a single NE. The value of *IPI* is made a configurable parameter which the operator can decide. Two minutes is assumed as the default value.

4.7 Summary

We have proposed techniques for controlling the trap arrival rate at the manager by monitoring the trap arrival rate from NEs and changing their trap forward type to "Pull" and "Push" accordingly. Techniques like distributed correlation and compression are proposed to limit the number of traps forwarded from NEs. A scheduling algorithm at the manager that gives a fair chance to all NEs for delivering their traps is also proposed. This algorithm also helps in fixing a time-limit for trap delivery which was not possible with the previously used trap delivery techniques. The values for the parameters used in these techniques/algorithms were also fixed from the analysis done on the field data obtained from different corDECT telecom networks.

CHAPTER 5

DESIGN AND IMPLEMENTATION

We have seen the proposed mechanisms for optimized trap delivery, the parameters involved in those techniques and fixed the values for those parameters in the previous chapter. This chapter gives an insight into the design and implementation aspects of the proposed mechanisms both at the manager and at the NE.

5.1 Design Issues

Apart from the different techniques discussed in the previous chapter, which are to be designed and developed, some important considerations for designing the complete solution are as follows.

5.1.1 Backward Compatibility

There are existing installations of NEs in the field. Not all of them would be upgraded at the same time. So, the manager software has to be designed in such a way that it provides backward-compatibility with the previous implementation of NEs. To distinguish old and new versions of NE a new attribute called "flow control supported" is added to attributes of NEs maintained by the manager. By default this parameter is set to false. When the NE is upgraded with the new flow control module, the value of this parameter is changed to "true" at the manager.

5.1.2 Consistency and Ease of Implementation

In order to ensure consistency and ease of implementation, the NE should use the same correlation rule set and correlation logic as used by the manager.

5.2 Correlation at the Manager

The basic requirement of correlation at manager is to present actual faults to the user without repetitions and clear them as and when they are cleared from the device. In order to correlate traps it is required that all the traps generated from the system are to be classified and assigned unique trapids. Then the relation between these traps should be established as a rule-set. For e.g, a rule could be trapid-1 clears trapid-2. When the traps are received the relationships that are identified as rules must be applied to them and it should be decided whether to present a particular trap to the user as fault or not. The database design and the logic to achieve this are given in this section.

5.2.1 Database Design for correlation

We have defined two tables that are used to store traps. One is the activealarms table which stores the outstanding problems in the network at any given instant. The traps from this table are only shown to the operator. The other table is the clearedalarms table. The traps that are cleared are moved to this table. As traps are moved frequently from activealarms table to clearedalarms table the same schema is maintained for both of them. The database schema for activealarms table along with the description of the fields is shown in Table 5-1. The rules to uniquely identify traps are maintained in a table called traptypeidlist. The schema of this table is shown in Table 5-2. The relationships between the traps are stored in traplookup table whose schema is presented in Table 5-3

Table 5-1 Database schema for activealarms

Column	Type	Modifiers	Description
recordid	bigint	not null	Unique number for trap
agentid	smallint	not null	The NE which generated this trap
moc	character varying(128)	not null	OID of Managed object class which generated this trap. E.g .1.3.6.1.2.1.2.2 for IFTABLE in MIB-II
moi	character varying(128)	not null	Managed object instance of the Managed object class which generated this trap.
eventtime	timestamp with time zone	not null	The time at which the trap was generated at the NE
loggingtime	timestamp with time zone	not null	The time at which the trap was logged at the manager
additionaltext	character varying(80)		A text as received from NE explaining the problem
eventtype	smallint	not null	The standard eventtype to which this trap belongs to
probablecause	smallint		The standard probable cause that further qualifies the eventtype
specificproblem	smallint		The device specific problem because of which this trap was generated
specificevent	smallint		A specific event affecting a MOC which has resulted in this trap
perceivedseverity	smallint	not null	The severity of the event which can take values of Critical, Major, Minor , Warning and Cleared
clearedmessage	character varying(80)		A text message which gives a reason for moving a failure trap from activealarms to clearedalarms.
clearedtime	timestamp with time zone		The time at which the failure trap was moved from activealarms to clearedalarms

traptypeid	smallint		The unique id that identifies trap of a particular type.
additionalattributes	character varying[]		Other attributes of interest obtained in the trap
correlationtype	smallint		The kind of correlation by which this trap has been cleared. The possible values are 1. System clear - This is a failure trap which is cleared by the system as the fault is rectified 2. Repeat - This is a repetitive trap. One such trap is already present in active trap list 3. Self cleared - This is a clear trap. This may clear other failure traps. 4. User cleared - This trap has been cleared manually by an operator at the management station
correlatedrecordids	bigint[]		For a System Clear trap, this field holds the recordid of the trap that cleared this failure trap. For a repeat trap, this field holds the recordid of the trap which is already present in active trap list. For a Self clear trap the recordid of itself is stored.
lastoccurrencetime	timestamp with time zone		The last time-stamp at which this trap was generated but was compressed
repeatcount	bigint		Count of number of times this trap has repeated between eventtime and lastoccurencetime
Indexes			
"activealarms_pkey" primary key, btree (recordid)			Index on recordid
"active_agentid_idx" btree (agentid)			Index on agentid
"active_traptypeid_idx" btree (traptypeid)			Index on traptypeid
"active_moi_idx" btree (moi)			Index on moi
Check constraints			
"activealarms_clearedmessage" CHECK (char_length(clearedmessage::text) <= 80)			Check to limit clear messages to 80 characters

Table 5-2 Datasbse schema for traptypeidlist

Column	Type	Modifiers	Description
traptypeid	smallint	not null	Unique identifier for the trap
moc	character varying	not null	OID for Managed object class. . E.g .1.3.6.1.2.1.2.2 for IFTABLE in MIB-II
classname	character varying	not null	Name of MOC as in MIB E.g, IFTABLE
eventtype	smallint		The standard eventtype to which this trap belongs to
probablecause	smallint		The standard probable cause that further qualifies the eventtype
specificproblem	smallint		The device specific problem because of which this trap was generated
perceivedseverity	smallint		The severity of the event which can take values of Critical, Major, Minor , Warning and Cleared
specificevent	smallint		A specific event affecting a MOC which has resulted in this trap
additionaltext	character varying(80)		A portion of the text as will be present in the trap
additionalattributes	character varying[]		Other attributes of interest in the trap and their values
Indexes:			
"traptypeidlist_pkey" primary key, btree (traptypeid)			Index on traptypeid

Table 5-3 Datasbse schema for traplookup

Column	Type	Modifiers
sourcetraptypeid	smallint	not null
action	character varying	not null
targettraptypeid	smallint	

5.2.2 Correlation Logic at the Manager

A PL/pgSQL procedure is written to perform correlation. The inputs to the procedure are recordid (R_{NEW}), NEID N from which the trap is received and the attributes received in the trap. The algorithm used in the stored procedure is given below:

Step 1. Assign trapid (TID_{SRC}) to incoming trap by comparing attributes in trap and traptypeidlist table entries

Step 2. Check if TID_{SRC} can clear any other trap by looking up traplookup table (i.e., to see if there is a record of type TID_{SRC} *"CLEARS"* TID_{TARGET} is present). If TID_{SRC} cannot clear any traps go to Step 6.

Step 3. Check if any trap with trapid TID_{TARGET} from the same NE and same MOI is present. If not go to Step 5. If found let us assume the recordid of that trap to be R_{OLD}

Step 4. Move trap with recordid R_{OLD} to clearedalarms. Update correlationtype as Systemcleared, Correlated recordid as R_{NEW} , clearedtime with current time and clearedmessage as "Correlated by system clear trap"

Step 5. Insert trap with recordid R_{NEW} into clearedalarms. Update correlationtype as Selfcleared, Correlated recordid as R_{NEW}, clearedtime with current time and clearedmessage as "Self cleared trap". Go to Step 9

Step 6. Check if a trap with trapid TID_{SRC} from NEID N, having the same MOI as the incoming trap is present in activealarms. If not go to Step 8. If found let us assume the recordid of that trap to be $R_{OLD-RPT}$

Step 7. Insert R_{NEW} into clearedalarms. Update correlationtype as Repeat, Correlated recordid as $R_{OLD-RPT}$, clearedtime with current time, clearedmessage as ‘Repeat trap” and set repeat count of $R_{OLD\text{-}RPT}$ as $R_{OLD\text{-}RPT}$ repeat count value + R_{NEW} repeat count value + 1. Go to Step 9

Step 8. Insert R_{NEW} into activealarms

Step 9. Exit from the procedure

In the correlation logic, the checks done to correlate and remove a trap because of ‘System Clear” or Self Clear” or ‘Repeat” is done in that order. This order was fixed based on the study of trap patterns found in corDECT systems. The indices used in the definition of the activealarms and clearedalarms tables aid in efficient execution of the stored procedure. It was found that on an average the execution time for the stored procedure was thirty-five to forty-milliseconds. Without the use of these indices, the execution time was more than five times higher at around two-hundred and fifty milliseconds.

5.3 Design for Controlling Traps Arrival Rate

On boot-up, the manager initializes from its database, the value for $TH_{TDM.}$ When a NE, which supports the new flow control mode, generates traps at a rate higher than the configured threshold in an *IPI* period, it sets the trap forward type of the NE to ‘Pull” mode. To indicate presence of traps the NE sends registration traps. The MIB attributes for sending this registration trap is given in Appendix A. Manager then periodically pulls traps from those NEs whose trap delivery mode is set to ‘Pull” and have sent registration trap. The number of traps obtained from a NE and the sum of repeat count in traps obtained from that NE in the current period is added up to arrive at the number of traps

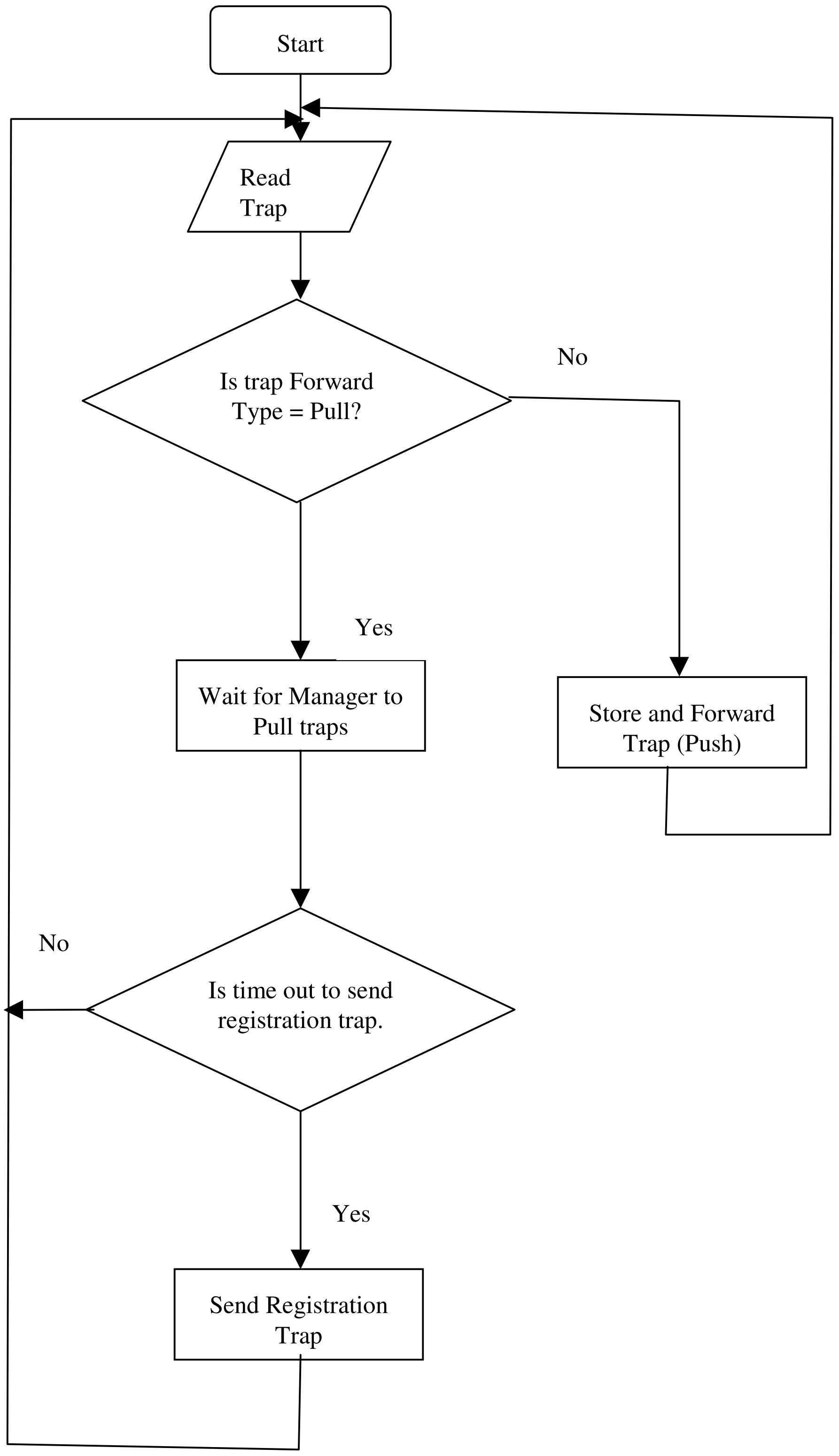

Start
Read Trap
Is trap Forward Type = Pull?
No
Yes
Wait for Manager to Pull traps
Store and Forward Trap (Push)
No
Is time out to send registration trap.
Yes
Send Registration Trap

Figure 5-1Design at agent for different values of trapForwardType generated from that NE in the current *IPI* period. If a NE whose trap delivery mode is set to 'Pull" generates traps at a rate which is less than TH_{TDM} for two consecutive *IPI*s then the trap delivery mode is changed to 'Push".

The behavior at the NE for different values of *trapForwardType* is shown in Figure 5.1

5.4 Design of Scheduling algorithm

At the start of every *IPI*, the scheduling algorithm is run to pick up the NEs from which traps will be pulled from. The next set of NEs to be pulled from is decided based on Equation [4.1]. For each NE whose trap forward type is set to 'Push" the following parameters are maintained.

1. Time of registration trap
2. Severity of traps pending to be delivered at NE as indicated by the registration trap
3. Number of times the NE was scheduled to be pulled from
4. Importance of the NE as configured by the operator

As an example, consider there are four NEs with the values of these parameters as shown in the Table 5-4. Further let us consider that only two out of these four NEs are to be scheduled. We will employ the scheduling algorithm to pick two NEs.

For each of these parameters, a rank is assigned. The ranks for the above parameters together with the final rank is shown in Table 5-5

Table 5-4 NEs to be scheduled

NEs	Severity	No. of Times Scheduled	Time of Registration	Importance
NE1	Critical	0	T1	IMPORTANT
NE2	Warning	0	T3	IMPORTANT
NE3	Minor	0	T2	IMPORTANT
NE4	Major	0	T4	IMPORTANT

Table 5-5 NEs merit list

NEs	Severity Rank	No. of Times Scheduled Rank	Time of Registration Rank	Importance Rank	Final Rank
NE1	1	1	1	1	4
NE2	4	1	3	1	9
NE3	3	1	2	1	7
NE4	2	1	4	1	8

From Table 5-5, it is seen that NEs with best ranks, viz., NE1 and NE3 will be scheduled for pulling traps. For the above illustration and for the experiments whose results are shown in Chapter-6, the function that calculates the final rank returned the sum of individual ranks for the parameters considered. It can be extended to consider different weights for different parameters, before arriving at the final figure of merit.

The overall flow at the manager at start of every IPI is shown in Figure 5.2

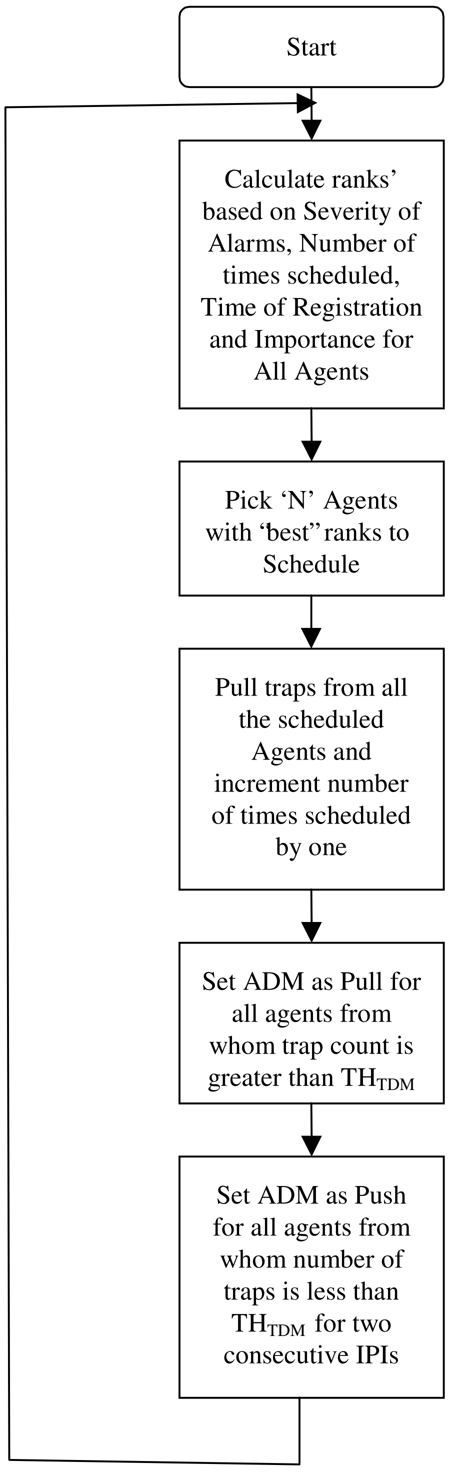

Start
Calculate ranks' based on Severity of Alarms, Number of times scheduled, Time of Registration and Importance for All Agents
Pick 'N' Agents with 'best" ranks to Schedule
Pull traps from all the scheduled Agents and increment number of times scheduled by one
Set ADM as Pull for all agents from whom trap count is greater than TH_{TDM}
Set ADM as Push for all agents from whom number of traps is less than TH_{TDM} for two consecutive IPIs

Figure 5-2 Overall design flow at manager at start of every IPI

5.5 Database design and Stored Procedure

To implement the scheduling algorithm, the information in registration traps from NEs and other statistics are stored in a database. The schema of the database table is shown below.

```
                 Table "controltraps"
          Attribute           |   Type   | Modifier
------------------------------+----------+-----------
 agentid                      | smallint | not null
 trapdeliverymode              | smallint | default 3
 importance                   | smallint | default 4
 numberofpendingtraps         | bigint   | default 0
 numberofpendingcritcaltraps  | bigint   | default 0
 numberofpendingmajortraps    | bigint   | default 0
 numberofpendingminortraps    | bigint   | default 0
 numberofpendingwarningtraps  | bigint   | default 0
 numberofregnrcvd             | bigint   | default 0
 regntraprcvdtime             | bigint   | default 0
 numberoftimesscheduled       | bigint   | default 0
 currentseverity              | smallint | default 0
 repeatcount                  | bigint   | default 0
 numberoftrapsinlastinterval  | bigint   | default 0
Index: controltraps_pkey
```

The value three for "trapdeliverymode" implies that the NEs will Push traps. The value four for "importance" implies that the NE is treated as very important.

When a registration trap is received from a NE, the contents of the "controltrap" table have to be updated. To do this a PL/pgSQL procedure was written. The procedure is given below.

```
create function updatecontroltraps (int2,int8,int8,int8,int8,int8,int8,int2 ) returns varchar
as '

DECLARE

      agntid alias for $1;

      numoftraps alias for $2;

      numofcritical alias for $3;

      numofmajor alias for $4;

      numofminor alias for $5;

      numofwarning alias for $6;

      registrationrcvdtime alias for $7;

      severity alias for $8;

      tmprow controltraps%rowtype;

      numofregistrationrcvd int;

      currsev int;

      PULL_TRAPS   int := 4;

      PUSH_TRAPS   int := 3;

begin

     raise notice ' '   agentid % ' '  ,agntid;

     select into tmprow * from controltraps where agentid = agntid;

     if not found then
```

```
            raise notice ' '   no such agent id %' '  ,tmprow.agentid;

            return null;

        end if;

-- increment number of times registration trap was received from this NE.

        numofregistrationrcvd = tmprow.numberofregistrationrcvd + 1;

        if severity < tmprow.currentseverity then

        currsev = severity;

        else

        currsev = tmprow.currentseverity;

        end if;

        if tmprow.regntraprcvdtime = 0 then

        -- registration trap from this NE is received for the first time. So store the
regntraprcvdtime obtained in the trap.

        raise notice ' '   updating registrationrcvd time ' '  ;

update controltraps set numberofpendingtraps=numoftraps,
numberofpendingcritcaltraps = numofcritical, numberofpendingmajortraps =
numofmajor, numberofpendingminortraps = numofminor ,
numberofpendingwarningtraps = numofwarning,numberofregistrationrcvd =
numofregistrationrcvd, regntraprcvdtime = registrationrcvdtime, currentseverity =
severity, trapdeliverymode = PULL_TRAPS where agentid = agntid;

        else

      -- registration trap has already been received from this NE. So preserve the
regntraprcvdtime and do not over write that column.

   raise notice ' '   not updating registrationrcvd time ' '  ;
```

```
    update controltraps set numberofpendingtraps = numoftraps, numberofpendingcritcaltraps = numofcritical, numberofpendingmajortraps = numofmajor, numberofpendingminortrps = numofminor, numberofpendingwarningtraps = numofwarning, numberofregistrationrcvd = numofregistrationrcvd, currentseverity = severity, trapdeliverymode = PULL_TRAPS where agentid = agntid;

    end if;

    return ' ' success' ' ;

end;

' language ' plpgsql' ;
```

5.6 Design of Correlation at the NE

The SAFT [11] at the NEs initializes the value for *LCEHT* from a configuration file. In addition it also reads two files named *traptypes.cfg* and *correlationrules.cfg*. The former contains the keys to be used to identify and associate a trap id to an incoming trap. The latter contains rules, which are used for correlating traps at the NE. It must be noted that these files are created by using simple SQL commands from the database tables at the manager, which are used for correlation.

To identify the trap id of an incoming trap various parameters are considered. The list of those parameters for couple of traps is given in Table 5-6. An incoming trap is scanned for the parameters listed in the *traptypes.cfg* file and the trap id of the trap is assigned. The *correlationules.cfg* file has rules that indicate the action to be taken on receipt of a trap. The format of this file with some examples is shown in Table 5-7

When a trap is generated at the NE, the trap id of the trap is found out by parsing the contents of the trap and comparing it with the values read from *traptypes.cfg* file. Let us name this trap T1 with trap id Id1. Then it is checked if this trap id is capable of clearing any traps by looking up the values read from *correlationrules.cfg* file. If the trap cannot clear any other trap, it is added to a linked list of traps and a timer is started for *LCEHT*. Within *LCEHT*, if another trap which clears T1 is received, then both T1 and the current trap are dropped. If till the expiry of *LCEHT*, no traps that clears T1 is obtained it is then forwarded to manager. Special care has to be taken before dropping traps at the NE. For example, consider the following rules in *correlationrules.cfg* as shown in Table 5-8. In this table we see that trap id 4 is capable of clearing traps with trap ids 6 and 8.

Table 5-6 Information in traptypes.cfg

Parameter	**Example Trap -1**	**Example Trap - 2**
Trap Id	1	2
Managed Object Class	CBS	CBS
Event type	Equipment Alarm	Equipment Alarm
Probable Cause	Equipment failure	Equipment failure
Specific Problem	Subsystem DOWN	Subsystem UP
Specific Event	Power problem	Power problem
Perceived Severity	Critical	Cleared
Additional Text	-	-
Additional Attributes	CBSSTATUS - 2	CBSSTATUS - 0

Table 5-7 Information in correlationrules.cfg

Source trap id	**Action**	**Target trap id**
2	CLEARS	1
4	CLEARS	6
10	DROP	

Table 5-8 Example of 1 trap clearing multiple traps

Source trap id	**Action**	**Target trap id**
4	CLEARS	6
4	CLEARS	8

Now consider the following sequence of events. At time t_1, trap id 8 is generated. It is forwarded to manager at t_1 + *LCEHT*. Now some time later at time t_2, trap id 6 is generated. At time t_3, which is less than t_2 + *LCEHT*, trap id 4 is generated. Since trap id 6 was not yet forwarded to manager, it would be correlated and removed. But it would not be correct to drop the trap with trap id 4, as it is needed at the manager to clear trap id 8, which happened at t_1 and has already been forwarded to manager. So traps that clear more than one trap cannot be dropped at the NE and have to be forwarded to manager.

As another example, consider that at time t_1, a trap T1 with trap id Id1 is generated and forwarded to manager as it could not be locally correlated and removed. Some time later, the NE reboots. Another trap T2 with Id1 is generated again after some time at time t_2.

And at time t_3 which is less than t_2 + *LCEHT*, a trap T3 with Id2 which is capable of clearing trap T2 with Id1 is generated. Now trap T2 with Id1 will be correlated and dropped. But it would not be correct to drop trap T3 with Id2 as it is required to clear trap T1 with Id1 at the manager. So in general, any trap that clears another trap can be dropped at NE only if it clears only one trap and the trap that it clears cannot occur twice without a clear trap in between under any circumstances.

5.7 Design of Data Structures for Correlation

To achieve the design described above, the data structures used should be chosen in such a way that it is optimized for searching. We have chosen hash table as the data structure for this.

5.7.1 Design of Data structure for traptypes.cfg

A two staged nested hash table is defined for this purpose. The contents of *traptypes.cfg* are read into a hash table. The key for this hash table is the Managed Object Class (MOC). The value is another hash table whose key is Event Type (ET) of the trap. The value of this second hash table has details like Probable Cause (PC), Specific Problem (SP), Specific Event (SE), Perceived Severity (PS), Additional Text (AT) and Additional Attributes (AA) that uniquely identify a trap. This arrangement is shown in the Figure 5.3.

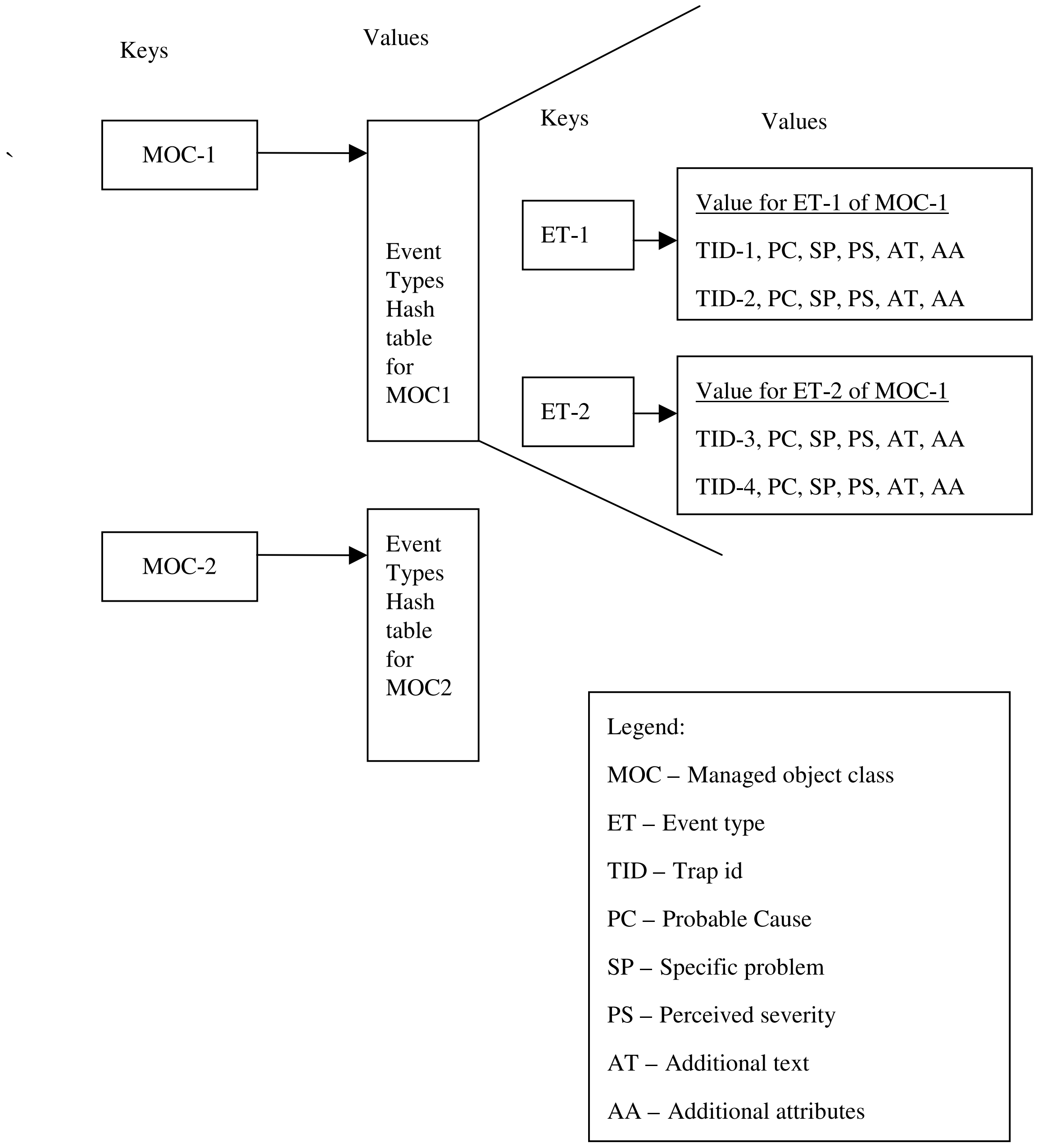

Keys
Values
MOC-1
Event Types Hash table for MOC1
Keys
Values
ET-1
Value for ET-1 of MOC-1
TID-1, PC, SP, PS, AT, AA
TID-2, PC, SP, PS, AT, AA
ET-2
Value for ET-2 of MOC-1
TID-3, PC, SP, PS, AT, AA
TID-4, PC, SP, PS, AT, AA
MOC-2
Event Types Hash table for MOC2
Legend:
MOC – Managed object class
ET – Event type
TID – Trap id
PC – Probable Cause
SP – Specific problem
PS – Perceived severity
AT – Additional text
AA – Additional attributes

Figure 5-3 Data structure organization for storing traptypes.cfg

This data structure reduces the number of searches needed to identify the Trap id. The corDECT system has thirty four distinct MOCs, eight distinct event types and three hundred and thirty seven distinct trap ids. The maximum number of trap ids for a MOC and for a given event type is seventy-five. So in a worst case scenario where there could be three hundred and thirty seven searches, in this hash table implementation there would be two lookups and linear search for seventy-five times which is seventy-seven percentage lesser number of searches.

5.7.2 Design of Data structure for correlationrules.cfg

The contents of *correlationrules.cfg* are read into a hash table. The key is the trap id and the value is an array in which each entry is an "action" and "target trap id" pair. This is shown in Figure 5.4. With this implementation, the searching operation required to check if an incoming trap can clear any other trap can be reduced to a lookup on this hash table. Once it is established that an incoming trap can clear other traps, the action to be taken is available in the array obtained by looking up the hash table. In the corDECT system there are three-hundred and sixty three correlation rules. As a worst case, it may take three-hundred and sixty three searches to find out if an incoming trap can clear any other trap. Because of this hash table implementation it will be reduced to a single look up. It must noted that finding out trap id of incoming trap and checking if the incoming trap can clear

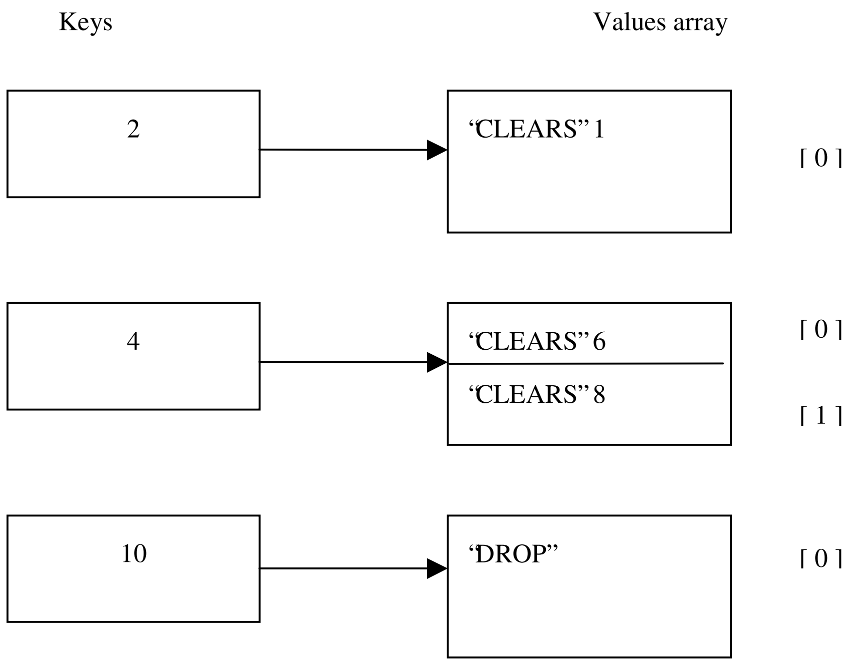


Figure 5-4 Data structure organization for storing correlationrules.cfg

any other trap is a operation that is to be repeated for each and every trap. So with these optimized data structures the operations are made faster and efficient.

5.8 Design of Compression at NE

The traps are stored in a file as commands that can be regenerated for transfer to manager. A Perl script is written to go through this file and identify traps that are same except for their time of generation. Once these identical traps are traced, the first occurrence of each such repetitive trap alone is retained. It is augmented with two more data like repeat count which carries the information of how many times this event occurred – say N – and the last occurrence time which is the generation time of the last of the identical traps. Now on receipt of this trap the manager can understand that the given event has occurred 'N" times between the generation time of the received trap and the last occurrence time as received in the trap. As an example consider the events shown in Table 5-9.

Table 5-9 Example of repetitive events

Time stamp	Trap Message
t1	File A – Read Failed
t2	File A – Read Failed
t3	File A – Read Failed
t4	File A – Read Failed

Table 5-10 Effect of compression

Time stamp	Trap Message	Additional Attributes
t1	File A – Read Failed	repeat count – 3, last occurrence time – t4

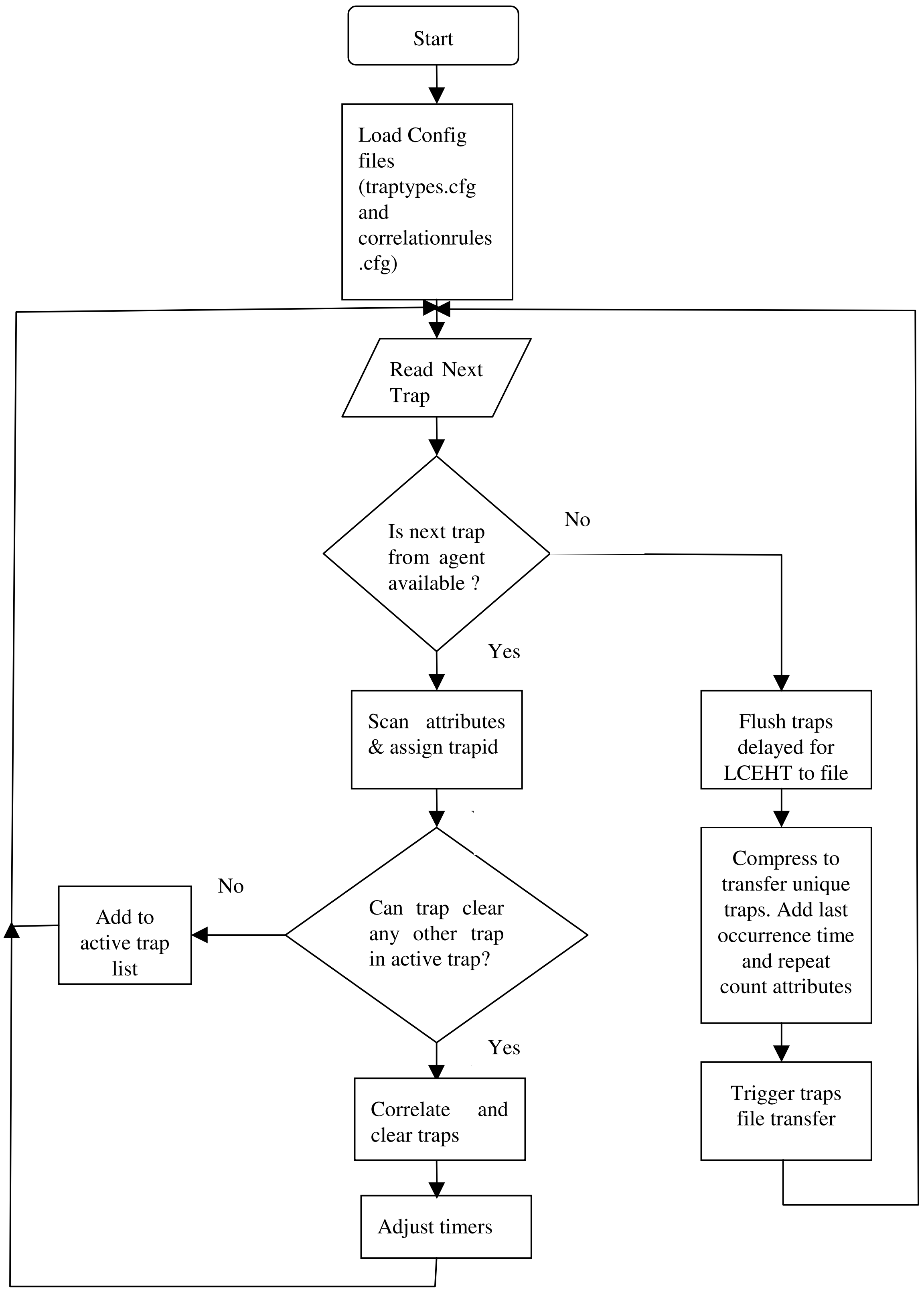

Start
Load Config files (traptypes.cfg and correlationrules .cfg)
Read Next Trap
Is next trap from agent available ?
No
Yes
Scan attributes & assign trapid
Flush traps delayed for LCEHT to file
Compress to transfer unique traps. Add last occurrence time and repeat count attributes
Trigger traps file transfer
Can trap clear any other trap in active trap?
No
Add to active trap list
Yes
Correlate and clear traps
Adjust timers

Figure 5-5 Correlation and Compression logic at NE

The trap forwarded to manager would be as shown in Table 5-10. So, instead of four traps, one trap with two additional attributes is forwarded to the manager. The logical flow at the NE for doing correlation and compression is shown in Figure 5.5.

5.9 Field trace Regenerator

To replay the events from a field scenario a field trace regenerator was designed and used. It is divided into two parts. The first part consists of a Perl module that reads the traps from the database dump obtained from the field and stores them in a file as commands that could be regenerated. The second part of the regenerator is another Perl script that goes through the file generated previously and generates the same traps at the same rate at which they were generated in the field by looking at successive time of generation of traps. For e.g, if there are traps at times *t1*, *t2*, *t3* and *t4*, after generating the trap *t1*, the script "sleeps" for a time t2-t1 before generating the next trap. This is used in the experiments done in lab to verify the implementation.

5.10 Summary

We have seen the design and implementation details of correlation at manager, changing the trap forward type from "Push" to "Pull" and vice-versa, the scheduling algorithm at the manager, the distributed correlation and compression logic at NEs. The database table used to store details of registration traps and the stored procedure used to update that table was presented. Use of optimized data structures for searching to perform correlation in an effective way was explained. Finally we have seen the field trace regenerator module which is used for simulating the traps at the same rate they were generated in the field. The experimental setup and results are shown in the next chapter.

CHAPTER 6

EXPERIMENTS AND RESULTS

In the previous chapter we have seen the design and implementation of the proposed mechanisms for optimized trap delivery. This chapter gives an insight into the simulation and lab experiments carried out to verify and vindicate the proposed mechanisms.

6.1 Experiment Setup

To aid in simulation, a field trace-driven experimental model was used. To evaluate the proposed mechanisms a set up as shown in Figure 6-1 was used.

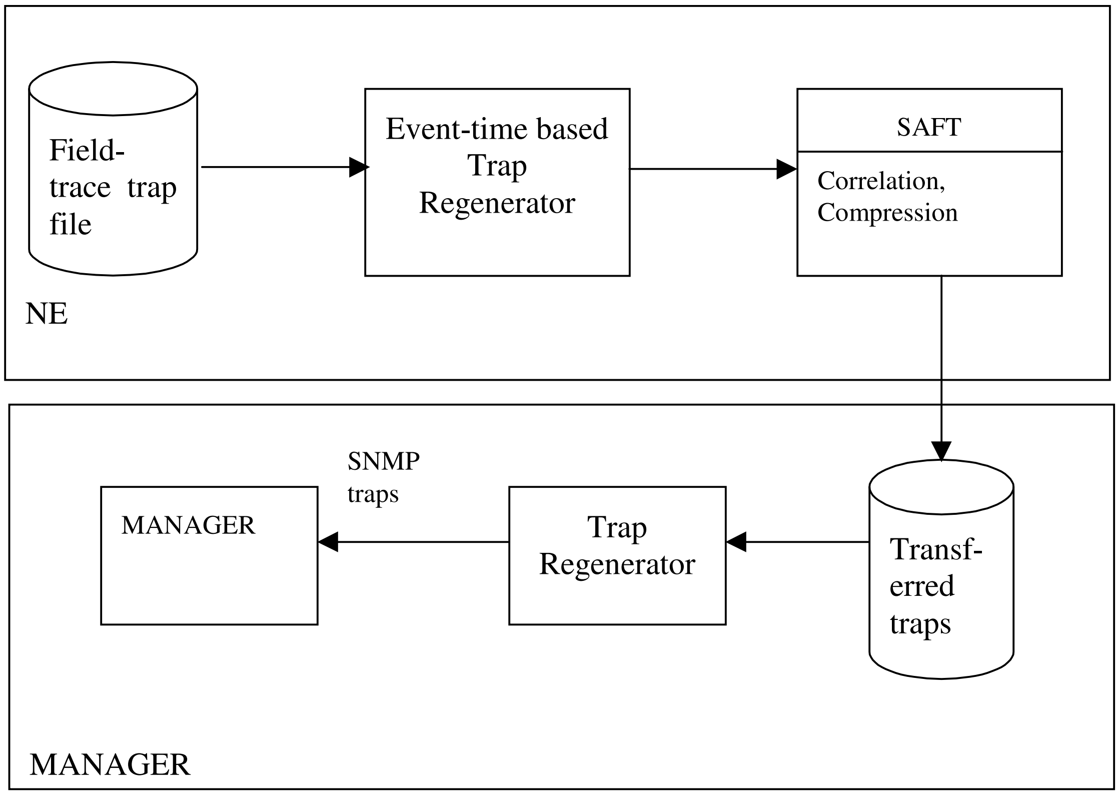


Figure 6-1 Experimental Setup

To verify the proposed mechanism, several 15-minute field traces when there was a flood of traps were considered and a 15-minute field trace from one of the NEs was chosen randomly during which 1594 traps were generated. This trace was replayed using the event-time based re-generator. For experiments with the proposed mechanisms, *LCEHT* was fixed at 10 seconds, TH_{TDM} was set at 4 traps, *IPI* was set to 2 minutes and N_{NE} was set to 10.

6.2 Experiment Results

With the experimental set up as described in the previous sections, readings were taken under following conditions

1. Existing mechanism
2. Proposed mechanism of flow control both at NE and manager, but only with compression at NE (No correlation)
3. Complete implementation of all proposed mechanisms both at NE and manager

In all conditions the following were noted down.

1. Number of traps received at the manager
2. Number of bytes transferred to the manager
3. Number of times traps were pushed to the manager
4. Number of times traps were pulled from the NE

These observations are plotted and shown in the following sections.

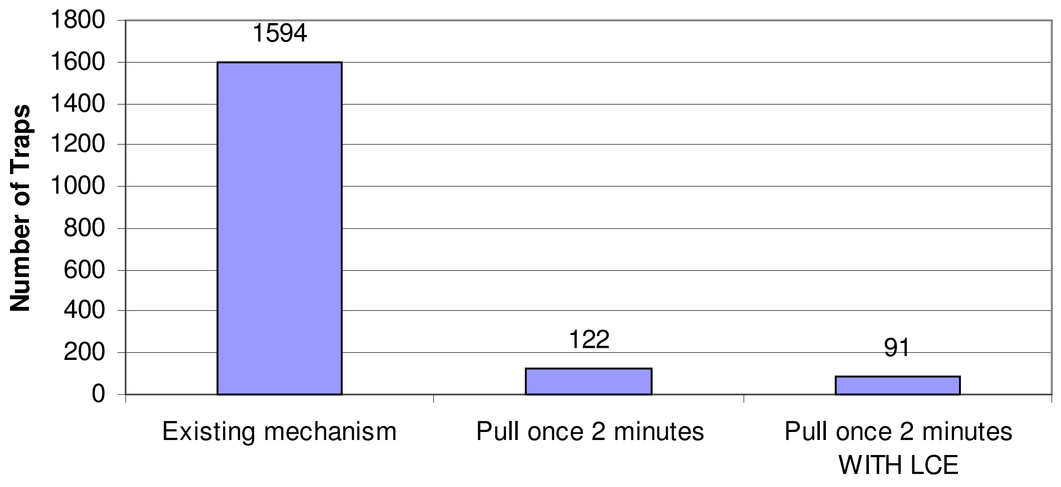


Figure 6-2 Comparison of number of traps received at manager

6.2.1 Comparison of number of Traps Received at Manager

Figure 6-2, shows the number of traps delivered in existing mechanism and in the proposed mechanism with and without LCE. It shows that the proposed mechanism with pull once in two minutes provides a gain of 92.34% and the proposed mechanism with Pull once in two minutes with LCE improves the gain to 94.29% in terms of number of traps delivered to the manager.

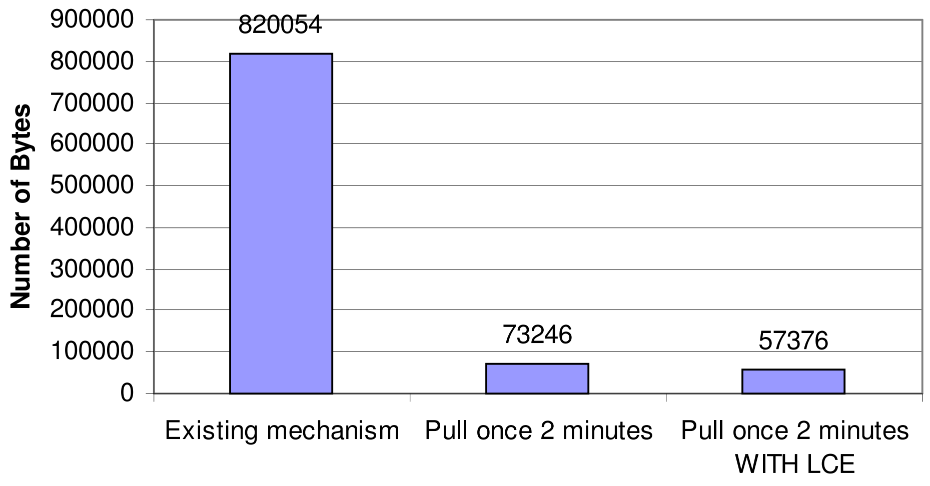


Figure 6-3 Comparison of number of bytes received at manager

6.2.2 Comparison of number of Bytes Transferred to Manager

Figure 6-3, shows the number of bytes transferred for delivering traps in existing mechanism and in the proposed mechanism with and without LCE. It shows that in the proposed mechanism with Pull alone, 91.06% lesser bytes are transferred when compared with the existing mechanism. This improves to 93% when LCE is included.

6.2.3 Effect of Push/Pull Mechanism

In the proposed mechanism, the NE pushes traps in 4 chunks. The controlling algorithm at the manager recognizes that the NE is flooding the manager with traps at a rate that is more than the TH_{TDM} and sets the trap forward type to ‘Pull” forbidding the NE to send traps. So, the load on the manager is reduced and the manager is able to control the arrival rate of traps. This is shown in Figure 6-4.

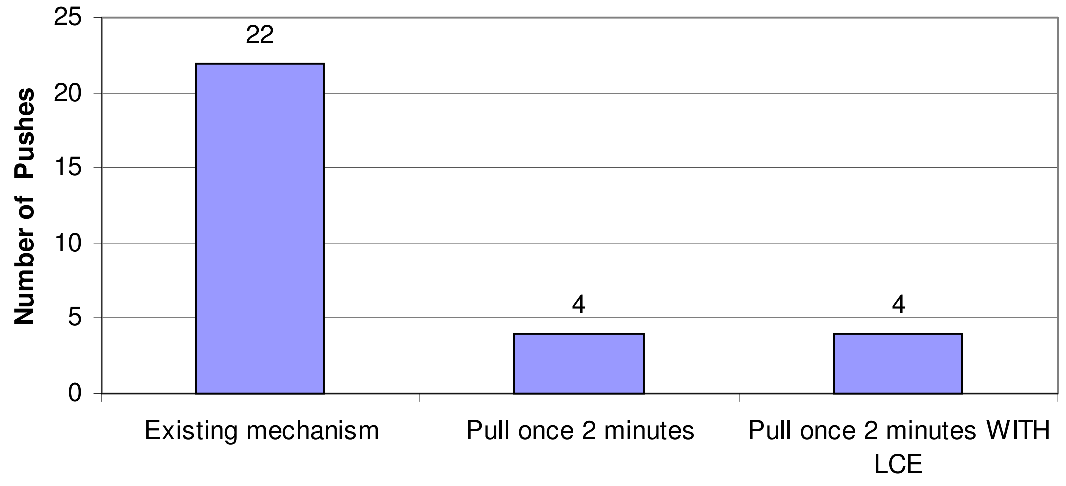


Figure 6-4 Comparison of number of pushes of traps into manager

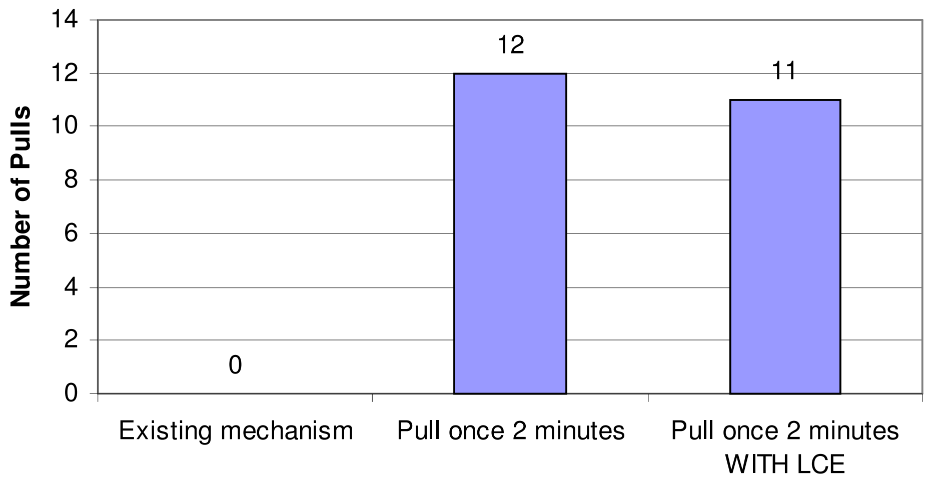


Figure 6-5 Comparison of number of pulls from NE

In the proposed mechanism, when the number of traps sent to manager becomes more than TH_{TDM}, the manager changes the trap delivery mode to "Pull" and periodically pulls from the NE. This is illustrated in Figure 6-5. Since in the existing mechanism there is no way to detect this flooding of traps, the manager PC will suffer a heavy CPU Utilization and CPU load.

6.2.4 Effect of LCE in Traps Forwarded from NE

The distributed correlation algorithm in place at the NE filters traps that are transient in nature. This is reflected in the Figure 6-6, which shows that out of 1594 traps generated, 61% of the traps were correlated and dropped at the NE.

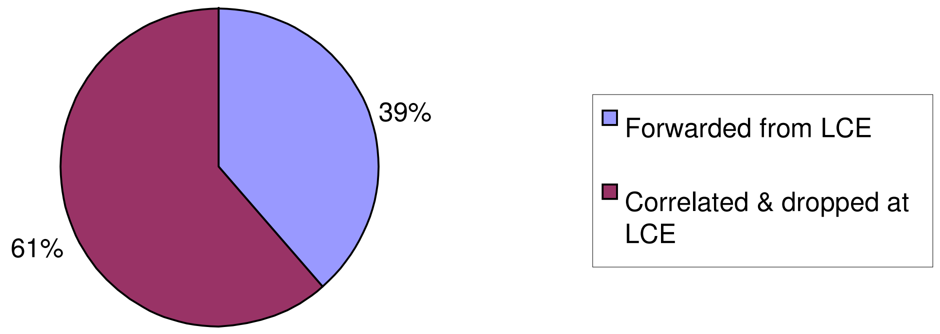


Figure 6-6 Effect of LCE

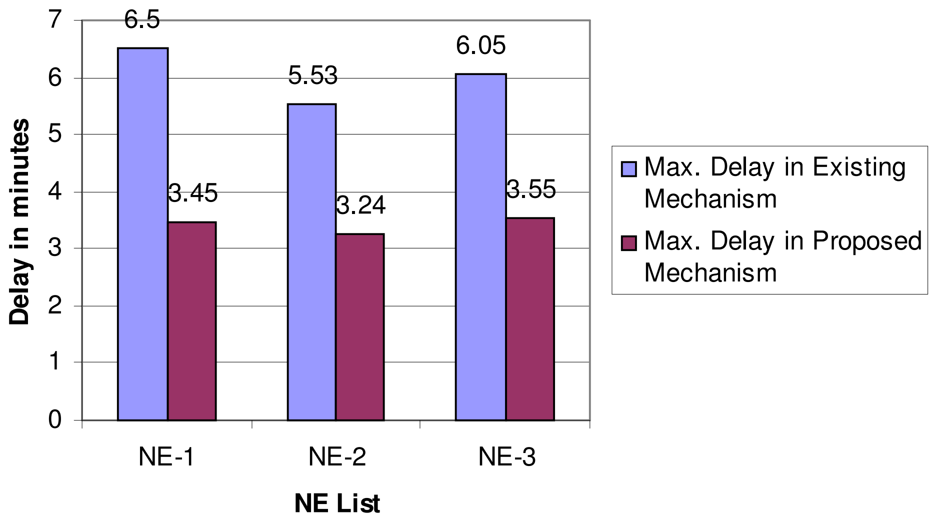


Figure 6-7 Comparison of delay in trap delivery

6.2.5 Delay in Trap Delivery

The delay in trap delivery in the existing mechanism was compared with the proposed mechanism. Traps were continuously generated from 3 NEs. As we see from Figure 6-7, the delay in proposed method is less than in the existing mechanism. An *IPI* of 2 minutes, *LCEHT* of 10 seconds and N_{NE} of 10 were used for the experiment. The average delay for the NEs was also found to be lesser in the proposed mechanism when compared with the existing mechanism.

6.2.6 QoS Time Limits

With a fair scheduling algorithm in place, it would now be possible to specify an upper bound on the trap delivery times for a given condition. For e.g, if there are 100 NEs, and

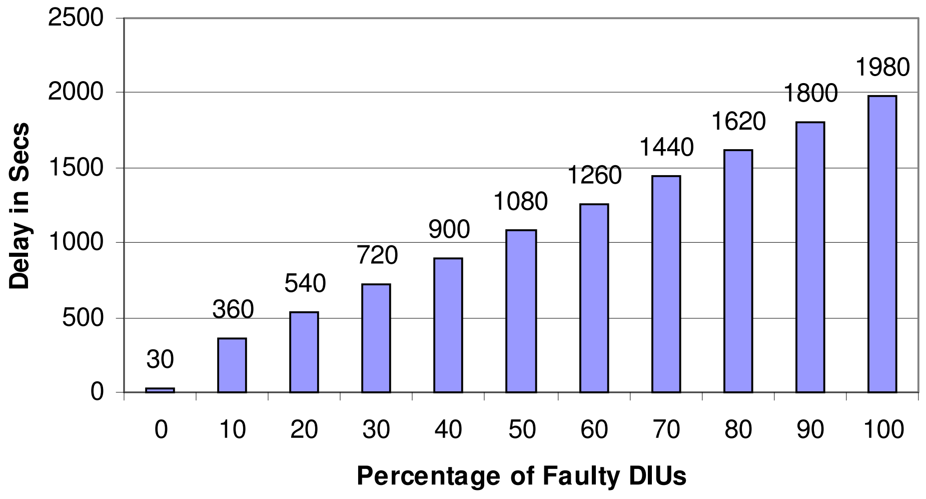


Figure 6-8 QoS time limits

IPI is 3 minutes, it is now possible to calculate the maximum time required for delivering traps. It is shown in Figure 6-8.

When there are no NEs that are faulty (i.e. they are generating traps at a rate less than AR_{HLT}), then traps would be pushed to manager, and they would be delivered in less than 30 seconds which is the typical time taken in SAFT-based trap transfer [11]. If 10% of the NEs are faulty then the scheduling algorithm would start to pull traps from these NEs one by one after 180 seconds and the traps from last NE would be pulled before 360 seconds i.e. before the start of the next *IPI*. Every 10% percent increase in percentage of faulty DIUs would increase the delay by another *IPI*. So even if 100% of the NEs are faulty, they would all be pulled from within 1980 seconds.

6.2.7 Sensitivity Analysis of the Algorithm

To test the algorithm under various conditions, the above set of experiments were repeated with different *IPI* values and by generating different number of traps. The *LCEHT* was fixed at 10 seconds, N_{NE} was fixed at 10 and the TH_{TDM} was fixed at 4 traps. In all the experiments, the number of traps received at manager, number of bytes transferred to manager, the number of Pushes into the manager and the numbers of Pulls from the NE were measured.

Figure 6-9 shows the number of traps received in existing mechanism and for various *IPI* values of the proposed mechanism. As the *IPI* increases, the number of traps received at the manager decreases. But it introduces a delay in the trap delivery time. Also it is interesting to note that the decrease in the number of traps for an increase in *IPI* above 4 minutes is not commensurate with the increase in *IPI*.

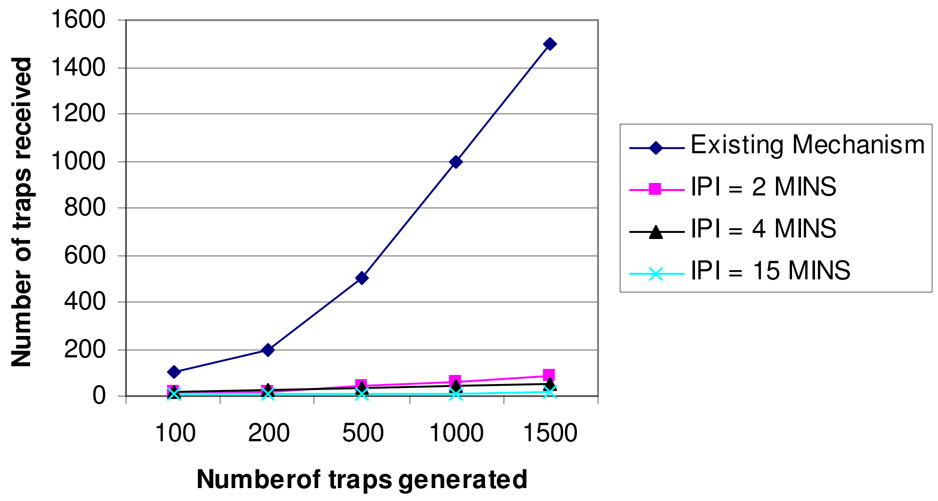


Figure 6-9 Comparison of number of traps received

Figure 6-10 shows the number of bytes transferred in transporting the traps in existing mechanism and for various *IPI* values of the proposed mechanism. As the *IPI* increases, the number of bytes transferred to the manager decreases. But it introduces a delay in the trap delivery time. Also it is interesting to note that the decrease in the number of bytes transferred to manager for an increase in *IPI* above 4 minutes is not commensurate with the increase in *IPI*. This vindicates the 2 minutes *IPI* chosen as default in Section 4.6.4

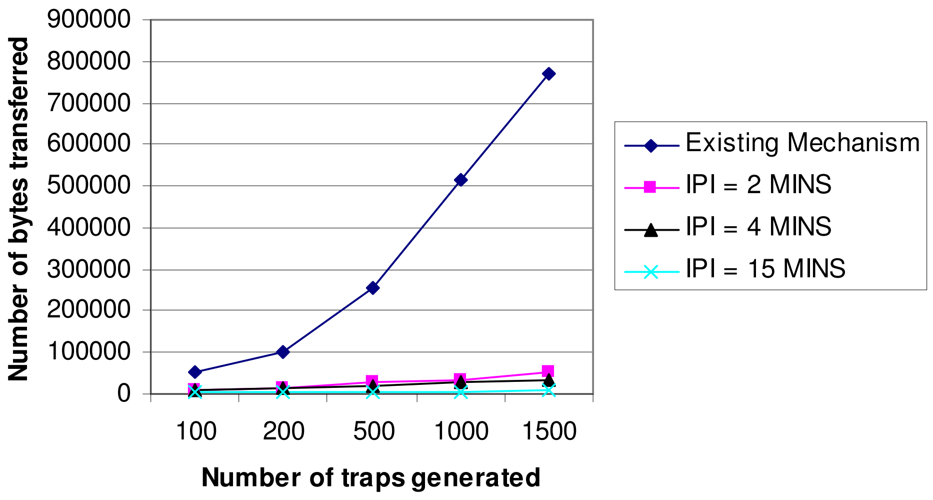


Figure 6-10 Comparison of number of bytes received

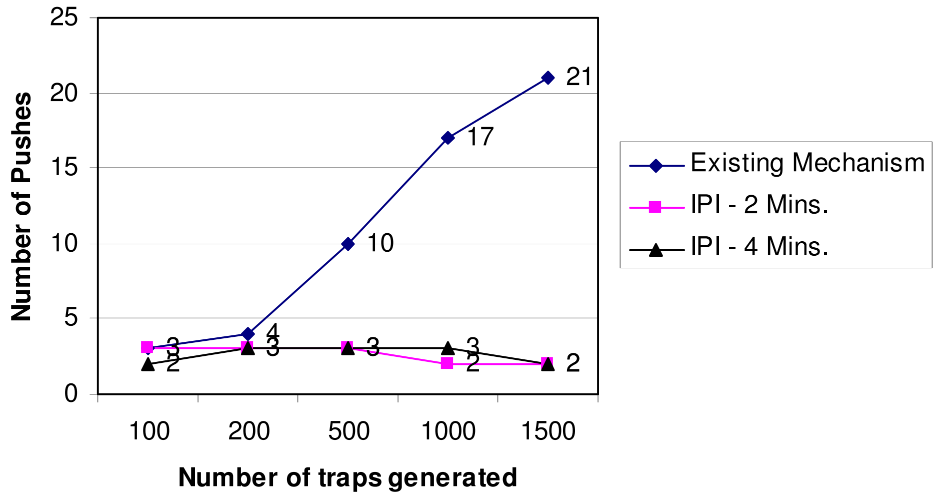


Figure 6-11 Comparison of number of Pushes into manager

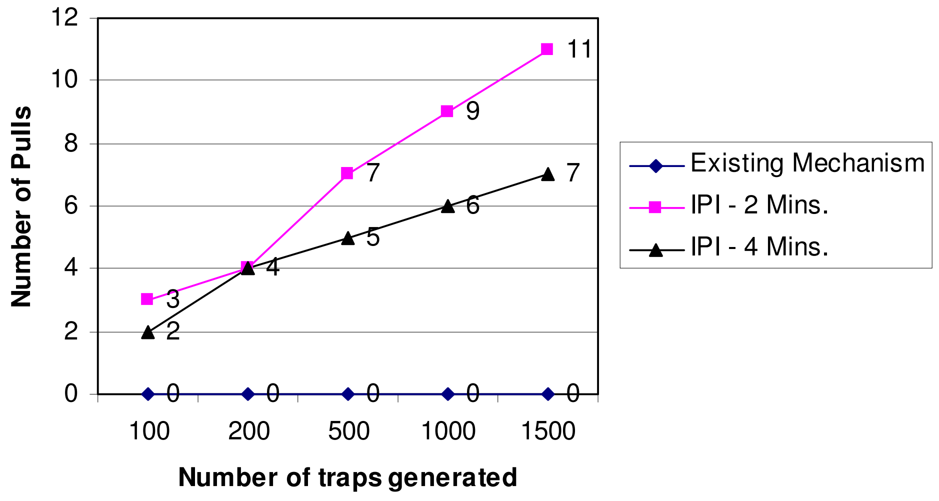


Figure 6-12 Comparison of number of Pulls from NE

In the proposed mechanism, the NE pushes traps for 2 to 4 times. The controlling algorithm at the manager recognizes that the NE is flooding the manager with traps at a rate that is more than the TH_{TDM} and sets the trap forward type to 'Pull" forbidding the NE to send traps. Because of lesser number of 'Push", the load on the manager is reduced and the manager is able to control the arrival rate of traps. This is shown in Figures 6-11 and 6-12. The number of Pulls decreases as the *IPI* increases.

6.2.8 Fairness of the Algorithm

To evaluate the fairness of the scheduling algorithm, a dry run of the same was done with the starting condition as shown in Table 6-1. The number of NEs to be scheduled in one run (N_{NE}) was fixed at 2 NEs. The dry run was performed with 10, 25, 50, 100 and 200 runs. In 10 runs all 20 NEs were scheduled exactly once and in 200 runs they were scheduled exactly 20 times each, showing the fairness of the algorithm. To evaluate the algorithm, when there is variation in the selection criteria, the initial condition was slightly changed so that one of the NEs – NE-4 – has a critical trap where as the other NEs have major trap pending to be delivered.

Table 6-1Condition before dry run of scheduling algorithm

NE	Severity	No. of times Sched.	Time of Registration	Importance
NE1	Critical	0	T1	IMPORTANT
NE2	Critical	0	T1	IMPORTANT
NE3	Critical	0	T1	IMPORTANT
NE4	Critical	0	T1	IMPORTANT
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
NE20	Critical	0	T1	IMPORTANT

Table 6-2 Condition before dry run of scheduling algorithm with severity of 1 NE higher than others

NE	Severity	No. of times Sched.	Time of Registration	Importance
NE1	Major	0	T1	IMPORTANT
NE2	Major	0	T1	IMPORTANT
NE3	Major	0	T1	IMPORTANT
NE4	Critical	0	T1	IMPORTANT
NE5	Major	0	T1	IMPORTANT
NE6	Major	0	T1	IMPORTANT
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
NE18	Major	0	T1	IMPORTANT
NE19	Major	0	T1	IMPORTANT
NE20	Major	0	T1	IMPORTANT

The initial condition before the dry run is shown in Table 6-2.

The dry run was again done with 10, 25, 50, 100 and 200 runs. The results of 10 and 200 runs are shown below.

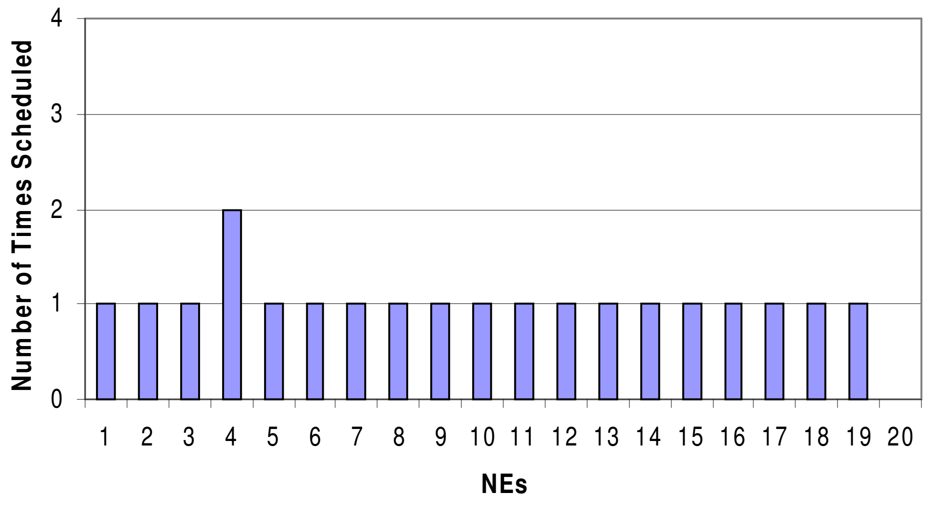


Figure 6-13 Scheduling distribution in 10 Runs with severity of 1 NE higher than others

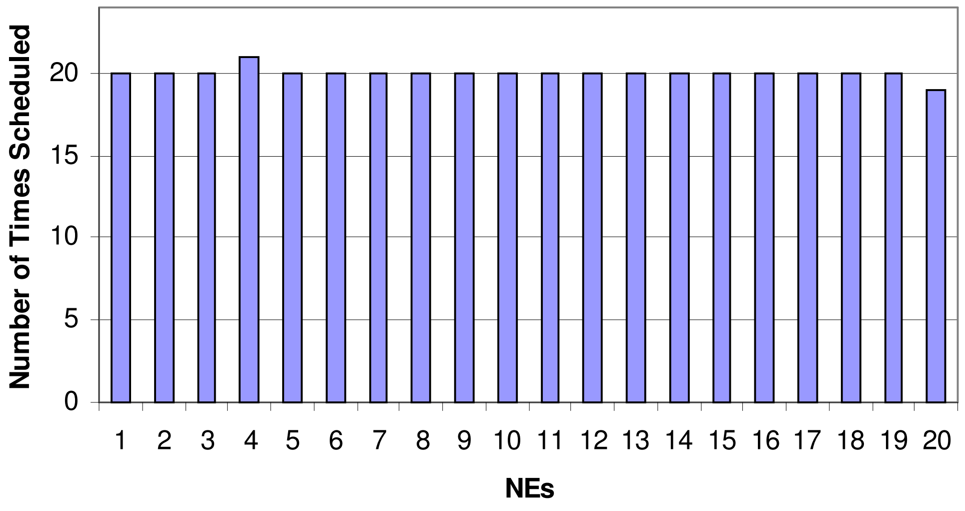


Figure 6-14 Scheduling distribution in 200 Runs with severity of 1 NE higher than others

The figures 6-13 and 6-14 show that the algorithm recognizes that NE-4 is to be given more chance than others because of a pending higher severity trap. The algorithm provides 2 and 21 chances to NE – 4 respectively in Figure 6-13 and 6-14. But at the same time it gives the other NEs also a fair chance to deliver their traps and is not overtly biased to NE – 4 which is a desirable feature.

6.2.9 Comparison of CPU Load and CPU Utilization

To compare the CPU load and utilization between the existing mechanism and the proposed mechanism, traps were generated continuously for one hour and the CPU load and CPU utilization were measured. Additionally, in the proposed mechanism, 10 NEs were scheduled in each *IPI* and the CPU load and utilization were measured. The findings are shown below. Figure 6-15 shows that the CPU utilization of the proposed mechanism with one NE being pulled from every *IPI* is 4 over an hour whereas the CPU utilization of the existing mechanism with one NE sending trap to manager ranges from 7 to 11. CPU utilization in the proposed mechanism with 10 NEs pulled from in every *IPI* is less than the CPU utilized in the existing mechanism with one NE sending traps.

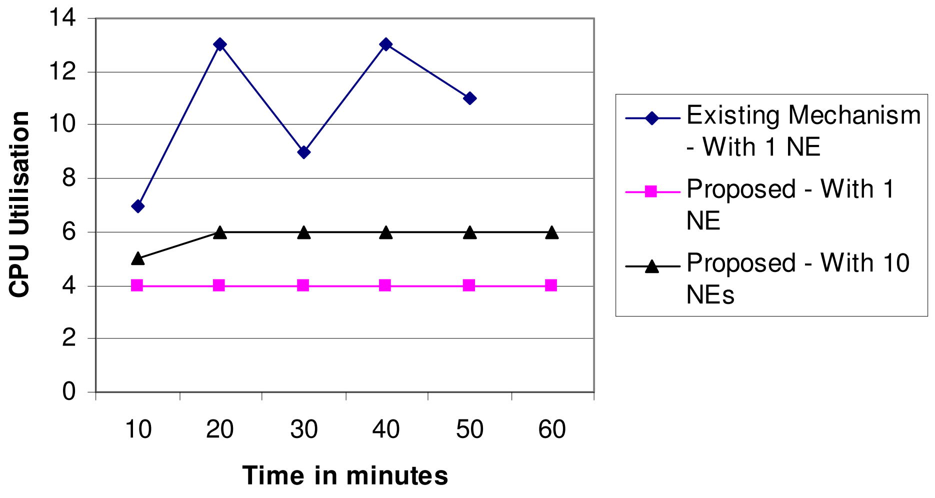


Figure 6-15 Comparison of CPU Utilization

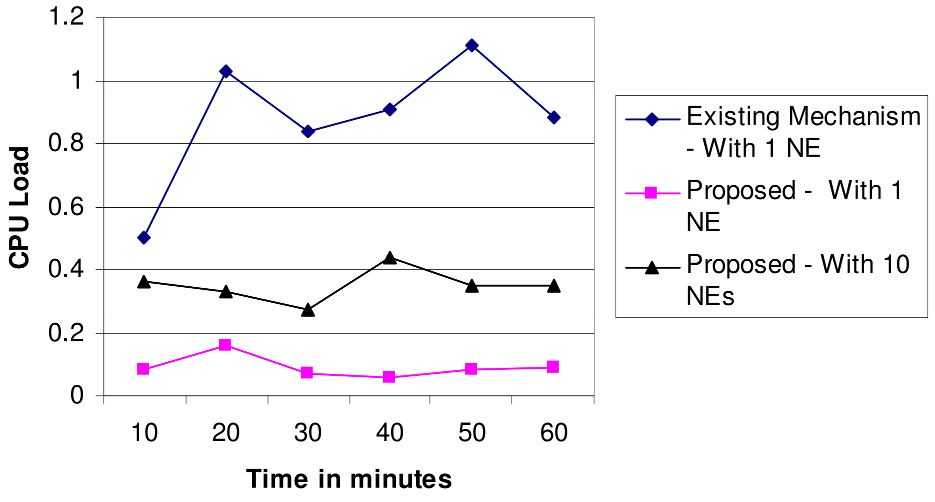


Figure 6-16 Comparison of CPU Load

Figure 6-16 shows that the CPU load of the proposed mechanism with one NE being pulled from every *IPI* ranges from range of 0.1 to 0.2 where as the CPU load of the existing mechanism with one NE sending trap to manager ranges from 0.5 to 1.1. Also it is observed that the CPU load in the proposed mechanism with 10 NEs being pulled from in every *IPI* is less than the CPU load in the existing mechanism with one NE sending traps.

As the figures 6-15 and 6-16 show, the proposed mechanism utilizes less CPU and causes lesser load on the system.

6.3 Summary

In this chapter we compared the proposed mechanisms against the existing mechanisms and saw that it came on top with better performances in terms of number of traps and bytes transferred, the CPU load and CPU utilization. We also saw that the scheduling algorithm provides a fair chance for all NEs to deliver the traps under different conditions.

CHAPTER 7

CONCLUSIONS

7.1 Summary and Conclusions

We have designed and developed a rule-based correlation engine to correlate traps and reduce the number of traps presented to the user. This mechanism is implemented and is in use in all corDECT deployments. It is seen from field data that this correlation logic is very effective and *less than one percent* of traps are being presented as faults to the operator.

We have proposed a mixture of Push and Pull based mechanisms for trap transfer to the manager. The proposed techniques give control to the manager to dynamically limit the trap arrival rate from each NE, limit the forwarding of traps from NEs by means of distributed correlation and compression. This was augmented with repeat count and last occurrence time to thwart loss of information. We have also proposed a scheduling algorithm that gives fair chance to all NEs to deliver traps. The scheduling algorithm helps us in providing QoS for traps by fixing an upper bound on time for delivering traps.

We have analyzed field data from different corDECT installations totaling 172 DIUs, serving nearly 2, 00,000 subscribers, for four weeks and used the results of the analysis to fix optimal values for the parameters that were part of the proposed techniques.

Implementation for the proposed techniques is done across various modules spawning NMS manager and NEs. The implementation is verified with the help of a field trace driven simulation. We compared the results with the existing mechanism and found that the proposed mechanism fared better in terms of number of traps, number of bytes transferred, and number of push of traps to the manager when compared to the existing mechanism. We also verified that the scheduling algorithm was fair to all NEs under different conditions. The proposed mechanism caused lesser CPU load and CPU utilization on the manager system.

The mechanisms proposed in this thesis are generic. Based on analysis of data of any network and the required QoS time limits, values for the parameters in the proposed mechanisms can be set to derive the aforementioned benefits

7.2 Future work

The values of the parameters involved in the proposed techniques like *IPI*, TH_{TDM}, N_{NE} and *LCEHT* can be calculated and varied dynamically to provide finer control and obtain better results.

Deploying the proposed mechanisms in field and gather data to get the real-world performance of the proposed mechanisms.

References

[1] A. Jhunjhunwala, B. Ramamurthi, T.A. Gonsalves, 'The role of technology in telecom expansion in India", *IEEE Commun. Mag* , vol. 22, no. 3, Nov. 98, pp. 88-94.

[2] A. Jhunjunwala, D. Jalihal, K. Giridhar, "Wireless in Local Loop – Some Fundamentals", *J. IETE*, vol. 46, no. 6, Nov – Dec 2000.

[3] ETSI EN 300 175-1, "Digital Enhanced Cordless Telecommunications (DECT) Overview", *European Telecomm. Stands. Inst.*, vol. 37, no. 8, Aug 89.

[4] Mani Subramanian*, "Network Management Principles and Practice"* Pearson Education, 2003, pp -40.

[5] CCITT Recommendation X.733, *Information Technology – Open Systems Interconnection - Systems Management: Part 4: Alarm reporting function*, 92.

[6] J. Case, M. Fedor, M. Schoffstall, J. Davin, A Simple Network Management Protocol (SNMP), RFC 1157, May 90.

[7] NET-SNMP http://net-snmp.sourceforge.net/ , Mar-2007

[8] J. Postal, User Datagram Protocol (IP), RFC 768, Aug. 80.

[9] J. Postal, Transmission Control Protocol (TCP), RFC 793, Sep. 81.

[10] L. Steinberg, Techniques for managing asynchronously generated alerts, RFC 1224, May 91.

[11] C.Jagadish et al, Low-cost data communication network for rural telecom network management, *Int .J. Network Mgmt*,2006, Wiley InterScience

[12] Data Communication Network, www.linuxports.com/howto/intro_to_networking/c563.htm, 2001.

[13] W. Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, 3rd ed., Addison Wesley, 99.

[14] A.S. Tanenbaum, *Computer Networks*, 4th ed., Prentice Hall of India, 2002.

[15] U. Black, *Network Management Standards: SNMP, CMIP, TMN, MIBs and Object Libraries*, 2nd ed., McGraw-Hill, 94.

[16] G. Goldszmidt and Y. Yemini, 'Delegated Agents for Network Management", *IEEE Commun. Mag*., vol.36, no. 3, Mar. 98, pp. 66 – 70.

[17] Banyan Networks, www.banyannetworks.com, 2004.

[18] K. Kant, M.M. Srinivasan, *Introduction to Computer System Performance Evaluation*, McGrawHill Inc., 92.

[19] L. Svododova, *Computer Performance Measurement and Evaluation Methods: Analysis and Applications*, American Elsevier publishing company, Inc., 76.

[20] A. Liotta and G. Pavlou, 'Exploiting Agent Mobility for Large-Scale Network Monitoring", *IEEE Network*, vol. 16, no. 3, May/Jun. 2002, pp. 7 – 15.

[21] T.A. Gonsalves, "The SNMP Manager-Agent paradigm revisited", *Proc. Intil.Conf. Broadband Commn. (ICBN '03)*, Banglore, India, May 2003.

[22] T.A. Gonsalves, A. Jhunhjunwala and H.A. Murthy, 'CygNet: Integrated network management for voice + internet", *Proc. NCC 2000*, IIT-Delhi, Jan. 2000.

[23] AGK Vanchynathan, N. Usha Rani, C. Charitha, T.A. Gonsalves, "Distributed NMS for Affordable Communications" *Proc. NCC 2004,* IIT Madras, 2004.

[24] TeNeT Group, IIT Madras, http://www.tenet.res.in , 2003

[25] Midas Communication Technologies Pvt. Ltd., http://www.midascomm.com, 2003.

APPENDIX - A MIB ATTRIBUTES FOR REGISTRATION TRAP

```
registrationTrapGroup             OBJECT IDENTIFIER ::= { corDECT 94 }

-- registrationTrapGroup starts here

registrationTrapTable OBJECT-TYPE
      SYNTAX      SEQUENCE OF RegistrationTrapEntry
      ACCESS      not-accessible
      STATUS      mandatory

        DESCRIPTION
            "The attributes of this table are used by manager to
            understand the details regarding the traps that are present
            in the NE"

      ::= { registrationTrapGroup 1 }

registrationTrapEntry   OBJECT-TYPE
      SYNTAX      RegistrationTrapEntry
      ACCESS      not-accessible
      STATUS      mandatory

      DESCRIPTION
                "RegistrationTrapEntry sequence for
RegistrationTrapTable"

      INDEX   { registrationTrapIndex }
      ::= { registrationTrapTable 1 }

RegistrationTrapEntry ::=
      SEQUENCE  {
      registrationTrapIndex                            TableIndex,
      registrationTrapNumOfPendingTraps                INTEGER,
      registrationTrapEventTime                        DisplayString,
      registrationTrapCumulativeSeverity               INTEGER,
      registrationTrapNumOfPendingCriticalTraps        INTEGER,
      registrationTrapNumOfPendingMajorTraps           INTEGER,
      registrationTrapNumOfPendingMinorTraps           INTEGER,
      registrationTrapNumOfPendingWarningTraps         INTEGER,
      registrationTrapNumOfPendingInfoTraps            INTEGER,
      registrationTrapNumOfRegnTrapsSent               INTEGER
}

registrationTrapRecordId             OBJECT-TYPE
        SYNTAX          TableIndex
        ACCESS          not-accessible
        STATUS          mandatory

        DESCRIPTION
                "Index for this table"
        ::=  {  registrationTrapEntry  1  }

registrationTrapNumOfPendingTraps  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                  "This attribute when present in a notification,
      indicates the number of traps which are yet to reach the
      management station"

        ::=  {  registrationTrapEntry  2  }

registrationTrapEventTime  OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                  "This attribute when present in a notification,
indicates the time at which the registration trap was generated"

        ::=  {  registrationTrapEntry  3  }

registrationTrapCumulativeSeverity       OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the highest severity among all  traps  which are yet to reach
the management station"

        ::=  {  registrationTrapEntry  4  }

registrationTrapNumOfPendingCriticalTraps   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the number of Traps with a severity of critical which are yet
to  reach the management station"
        ::=  {  registrationTrapEntry  5  }

registrationTrapNumOfPendingMajorTraps   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the number of Traps with a severity of major which are yet to
reach the management station"
```

```
        ::=  {  registrationTrapEntry  6  }

registrationTrapNumOfPendingMinorTraps   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the number of Traps with a severity of minor which are yet to
reach the management station"

        ::=  {  registrationTrapEntry  7  }

registrationTrapNumOfPendingWarningTraps   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the number of Traps with a severity of warning which are yet
to reach the management station"

        ::=  {  registrationTrapEntry  8  }


registrationTrapNumOfPendingInfoTraps        OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the number of traps with no severity field which are yet to
reach the management station"

        ::=  {  registrationTrapEntry  9  }

registrationTrapNumOfRegnTrapsSent  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory

        DESCRIPTION
                "This attribute when present in a notification,
indicates the number of times the registration trap was sent to the
management station indicating the availability of traps to be
delivered"

        ::=  {  registrationTrapEntry  10  }


-- registrationTrapGroup ends here
```