Simulation Of Events in WiMAX Element Management System and Optimize communication

PREPARED BY :

Jigar Shah
09MCE014

GUIDED BY :

Mr. Thomas Nishanth

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY
AHMEDABAD

Simulation of Events in WiMAX Element Management System and Optimize communication

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By
Jigar Shah
(09MCE014)

Guided By
Mr. Thomas Nishanth

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AHMEDABAD-382481

May 2011

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Computer Science and Engineering at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Jigar Shah

Certificate

This is to certify that the Major Project entitled "Simulation of Events in WiMAX Element Management System and Optimize communication" submitted by Jigar Shah (09MCE014), towards the fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

K. S. Raju,
Manager,
R&D Center,
Motorola,
Hyderabad.

Dr.S.N.Pradhan,
Professor and PG-Coordinator,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad.

Prof.D.J.Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad.

Dr.K.Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad.

Abstract

An Element management system (EMS) consists of systems and applications that are concerned with managing network elements (NE) on the network element management layer (NEL) of the Telecommunication Management Network model.

As recommended by ITU-T, the Element Management System's key functionality is divided into five key areas - Fault, Configuration, Accounting, Performance and Security (FCAPS). Portions of each of the FCAPS functionality fit into the TMN models. On the northbound the EMS interfaces to Network Management Systems and or Service Management Systems depending on the deployment scenario. Southbound the EMS talks to the devices.

This dissertation addresses two key factors in WiMAX-EMS, namely enhance the performance of WiMAX-EMS and improve the quality. First part, improve quality, deals with removing loop holes.Second part, enhance performance deals with multithreading concept and synchronization problem between EMS and NE, which are communicating through SNMP.

Acknowledgements

My sincere thanks to the fine people around me who helped me in completing this project work. Their wisdom, clarity of thought and support motivated me to bring this project to its present state. First, I wish to thank Mr. Vishwanath Kotta (Senior Manager) for giving me an opportunity to work on this project. His continued support, guidance and vision have helped me in this project, and it has truly been a pleasure working with him.

My heartfelt thanks go to Mr. K Raju, Mr. Nishanth Thomas and my team members for their invaluable guidance, which not only enabled me to sort out the technical issues but also helped me in updating my knowledge, which undoubtedly will also be useful in the future.

I wish to place on record my gratitude to Motorola, Hyderabad for providing me an opportunity to work with them on this project of such importance. My stay in the organization has been a great learning experience and a curtain raiser to an interesting and rewarding career. This exposure has enriched me with technical knowledge and has also introduced me to the attributes of a successful professional.

I wish to express my deep gratitude towards Dr. S.N. Pradhan (PG Coordinator) and all professors at Nirma Institute of technology who taught the fundamental essentials to undertake such a project. Without their valuable guidance it would have been extremely difficult to grasp and visualize the project theoretically.

Finally, I would like to thank my parents and friends for their constant love and support and for providing me with the opportunity and the encouragement to pursue my goals.

- Jigar Shah
09MCE014

Contents

List of Figures

1

Project Introduction

1.1 Introduction

EMS (Element Management System)

- An Element management system (EMS) consists of systems and applications that are concerned with managing network elements (NE) on the network element management layer (NEL) of the Telecommunication Management Network model.
- As recommended by ITU-T, the Element Management System's key functionality is divided into five key areas - Fault, Configuration, Accounting, Performance and Security (FCAPS). Portions of each of the FCAPS functionality fit into the TMN models. On the northbound the EMS interfaces to Network Management Systems and or Service Management Systems depending on the deployment scenario. Southbound the EMS talks to the devices.
- An element management system (EMS) manages one or more of a specific type of telecommunications network element (NE). Typically, the EMS manages the functions and capabilities within each NE but does not manage the traffic between different NEs in the network. To support management of the traffic between itself and other NEs, the EMS communicates upward to higher-level network management systems (NMS) as described in the telecommunications management network (TMN) layered model.

1.2 Objective

Simulation of events to verify EMS functionality (For Fault management module) and improve and optimize FM (Fault Management)module in a way that performance and quality of FM module increased

1.3 Scope of work

As the title suggests the goal of this project, work carried out in this research is useful in improving performance and quality of the Element Management system.

This project is divided into two goal, one improve the quality of the EMS by removing the bottleneck through simulation of Events of FM module and second, enhance the performance of FM module by applying multithreading concept and solving synchronization problem in SNMP protocol.

1.4 Motivation

WiMAX-EMS is used to monitor more than 1000 Network element. So it is expected to consume less time to process events for all network elements as it is working with more than 1000 network elements. EMS again has to work with NMS its parent system so it should always be optimized for consuming less time.

1.5 Thesis Organization

The rest of the thesis is organized as follow.

Chapter 2, Literature Survey, includes the overview of WiMAX-EMS and SNMP protocol.

Chapter 3, Problem Definition, includes problem regarding quality and performance.

Chapter 4, Technology, solution and methodology, includes solution and methodology of the problem. And technology used to solve problem.

Chapter 5, Implementation and Result, includes implementation of solution and result of that solution.

Chapter 6, Conclusion and Future work, concludes this project with a summary, and provides possible directions for relevant future research.

2

Literature Survey

2.1 EMS (Element Management System)

- An Element management system (EMS) consists of systems and applications that are concerned with managing network elements (NE) on the network element management layer (NEL) of the Telecommunication Management Network model.
- As recommended by ITU-T, the Element Management System's key functionality is divided into five key areas - Fault, Configuration, Accounting, Performance and Security (FCAPS). Portions of each of the FCAPS functionality fit into the TMN models. On the northbound the EMS interfaces to Network Management Systems and or Service Management Systems depending on the deployment scenario. Southbound the EMS talks to the devices.
- An element management system (EMS) manages one or more of a specific type of telecommunications network element (NE). Typically, the EMS manages the functions and capabilities within each NE but does not manage the traffic between different NEs in the network. To support management of the traffic between itself and other NEs, the EMS communicates upward to higher-level network management systems (NMS) as described in the telecommunications management network (TMN) layered model.

`Position of EMS in Network Architecture`
As shown in figure, NEs in the network each communicate with their respective EMS. The NE specific EMSs communicate via either proprietary or, preferably, an open, standard, northbound interface to a higher level NMS that provides integrated multivendor network management.

`Networks to be managed`
Devices, systems and/or anything else requiring some form of monitoring and management

- Telecommunications equipment
  Telecom switches, BTS, BSC, trunks, subscriber lines, etc.
- Data Networks
  LANs, WANs, Modems, Repeaters, Hubs, Bridges, etc.

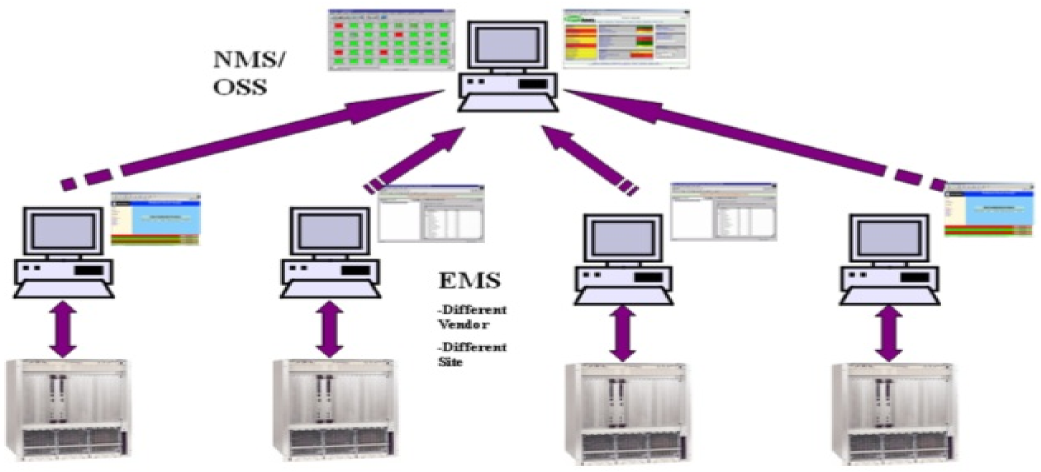


Figure 2.1: Position of EMS in Network Management

2.2 Management function area (FCAPS)

Fault : RAS Quality Assurance, Alarm Surveillance, Fault Localization, Fault Correction, Testing, Trouble Administration

Configuration : Network Planning and Engineering, Installation, Service Planning and Negotiation, Provisioning, Status and Control

Accounting : Pricing, Usage Measurement, Collections and Finance, and Enterprise Control

Performance : Quality Assurance, Performance Monitoring, Performance Control, and Performance Analysis

Security : Prevention, Detection, Containment and Recovery, and Security Administration

- Portions of each of the FCAPS functionality will be performed at different layers of the TMN architecture. As an example, fault management at the EML is detailed logging of each discrete alarm or event. The EMS then filters the alarms and forwards them to an NMS that perform alarm correlation across multiple nodes and technologies to perform root cause analysis. A subset of FCAPS functionality is listed in Table.

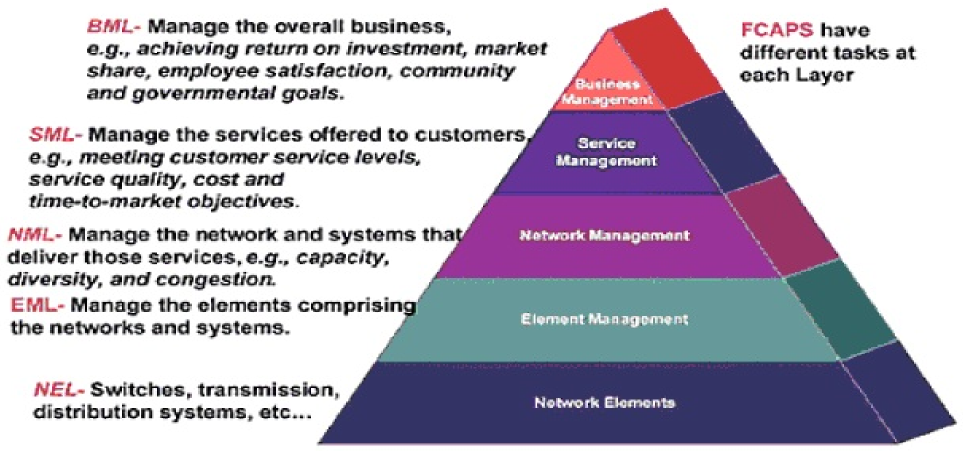
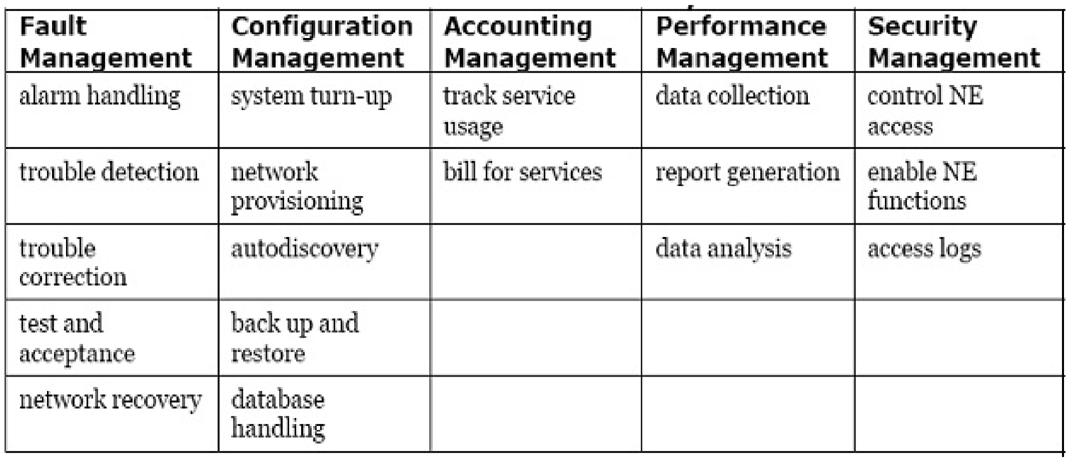

BML- Manage the overall business, e.g., achieving return on investment, market share, employee satisfaction, community and governmental goals.
SML- Manage the services offered to customers, e.g., meeting customer service levels, service quality, cost and time-to-market objectives.
NML- Manage the network and systems that deliver those services, e.g., capacity, diversity, and congestion.
EML- Manage the elements comprising the networks and systems.
NEL- Switches, transmission, distribution systems, etc...
Business Management
Service Management
Network Management
Element Management
Network Elements
FCAPS have different tasks at each Layer

Figure 2.2: Five layer TMN Network Management Architecture



Fault Management	Configuration Management	Accounting Management	Performance Management	Security Management
alarm handling	system turn-up	track service usage	data collection	control NE access
trouble detection	network provisioning	bill for services	report generation	enable NE functions
trouble correction	autodiscovery		data analysis	access logs
test and acceptance	back up and restore			
network recovery	database handling			

Figure 2.3: Table 1. A Subset of FCAPS Functionality

2.2.1 Fault Management

In network management, Fault Management is the set of functions that detect, isolate, and correct malfunctions in a WiMAX-EMS, compensate for environmental changes, and include maintaining and examining error logs, accepting and acting on error detection notifications, tracing and identifying faults, carrying out sequences of diagnostics tests, correcting faults, reporting error conditions, and localizing and tracing faults by examining and manipulating database information. When a fault or event occurs, a network element will often send a notification to the network operator using a protocol such as SNMP. An alarm is a persistent indication of a fault that clears only when the triggering condition has been resolved.

FM Alarm Life Cycle

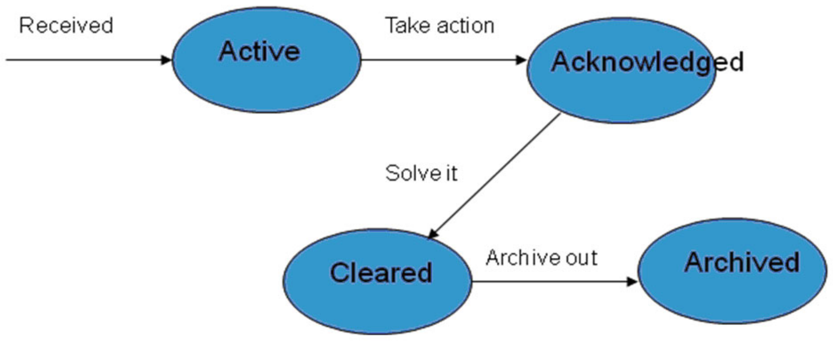
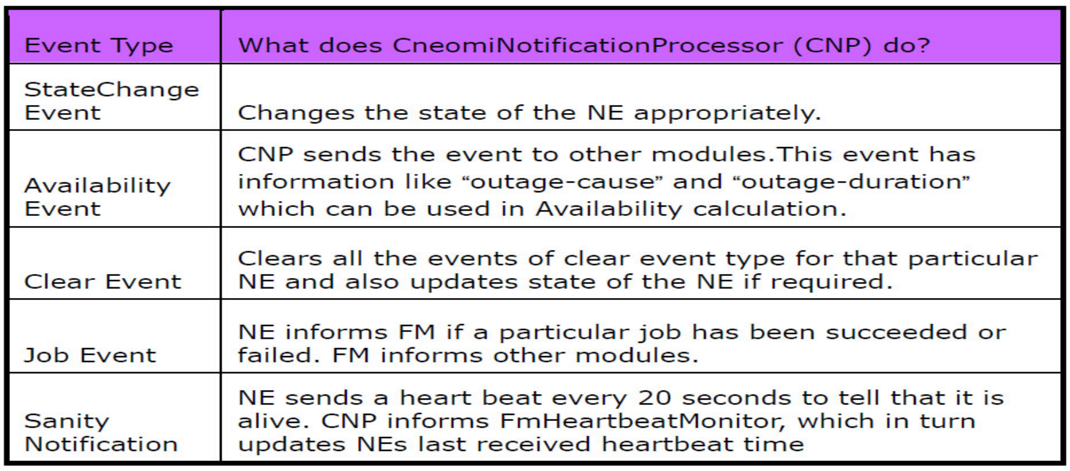
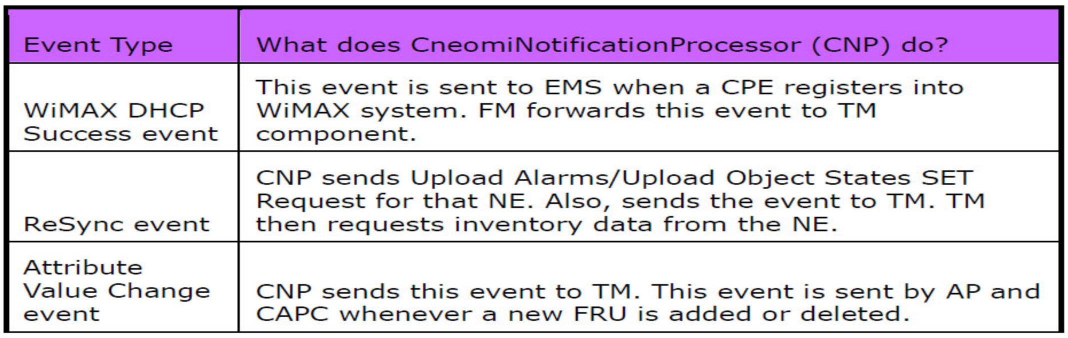


Figure 2.4: FM Alarm Life cycle

Types of Events



Event Type	What does CneomiNotificationProcessor (CNP) do?
StateChange Event	Changes the state of the NE appropriately.
Availability Event	CNP sends the event to other modules.This event has information like “outage-cause” and “outage-duration” which can be used in Availability calculation.
Clear Event	Clears all the events of clear event type for that particular NE and also updates state of the NE if required.
Job Event	NE informs FM if a particular job has been succeeded or failed. FM informs other modules.
Sanity Notification	NE sends a heart beat every 20 seconds to tell that it is alive. CNP informs FmHeartbeatMonitor, which in turn updates NEs last received heartbeat time



Event Type	What does CneomiNotificationProcessor (CNP) do?
WiMAX DHCP Success event	This event is sent to EMS when a CPE registers into WiMAX system. FM forwards this event to TM component.
ReSync event	CNP sends Upload Alarms/Upload Object States SET Request for that NE. Also, sends the event to TM. TM then requests inventory data from the NE.
Attribute Value Change event	CNP sends this event to TM. This event is sent by AP and CAPC whenever a new FRU is added or deleted.

2.3 SNMP

- Simple Network Management Protocol (SNMP) is a UDP-based network protocol. It is used mostly in network management systems to monitor network attached devices for conditions that warrant administrative attention. SNMP is a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF). It consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects.
- SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications.

2.3.1 Overview and basic concepts

In typical SNMP use, one or more administrative computers called managers have the task of monitoring or managing a group of hosts or devices on a computer network. Each managed system executes, at all times, a software component called an agent which reports information via SNMP to the manager. Essentially, SNMP agents expose management data on the managed systems as variables. The protocol also permits active management tasks, such as modifying and applying a new configuration through remote modification of these variables. The variables accessible via SNMP are organized in hierarchies. These hierarchies, and other metadata (such as type and description of the variable), are described by Management Information Bases(MIBs).

An SNMP-managed network consists of three key components:

- Managed device
- Agent software which runs on managed devices
- Network management system (NMS) software which runs on the manager

A managed device is a network node that implements an SNMP interface that allows unidirectional (read-only) or bidirectional access to node-specific information. Managed devices exchange node-specific information with the NMSs. Sometimes called network elements, the managed devices can be any type of device, including, but not limited to, routers, access servers, switches, bridges, hubs, IP telephones, IP video cameras, computer hosts, and printers.

An agent is a network-management software module that resides on a managed device. An agent has local knowledge of management information and translates that information to or from an SNMP specific form.

A network management system (NMS) executes applications that monitor and control managed devices. NMS's provide the bulk of the processing and memory resources required for network management. One or more NMSs may exist on any managed network.

2.3.2 Management Information Base (MIB)

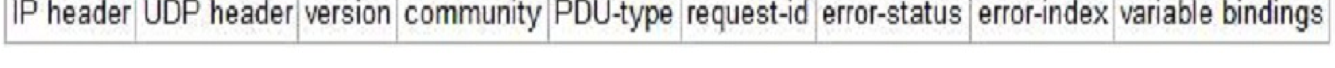
SNMP itself does not define which information (which variables) a managed system should offer. Rather, SNMP uses an extensible design, where the available information is defined by management information bases (MIBs). MIBs describe the structure of the management data of a device subsystem; they use a hierarchical namespace containing object identifiers (OID). Each OID identifies a variable that can be read or set via SNMP. MIBs use the notation defined by ASN.1.

2.3.3 Protocol Details

SNMP operates in the Application Layer of the Internet Protocol Suite (Layer 7 of the OSI model). The SNMP agent receives requests on UDP port 161. The manager may send requests from any available source port to port 161 in the agent. The agent response will be sent back to the source port on the manager. The manager receives notifications (Traps and InformRequests) on port 162. The agent may generate notifications from any available port.

SNMPv1 specifies five core protocol data units (PDUs). Two other PDUs, GetBulkRequest and InformRequest were added in SNMPv2 and carried over to SNMPv3.

All SNMP PDUs are constructed as follows:



IP header	UDP header	version	community	PDU-type	request-id	error-status	error-index	variable bindings

Figure 2.5: SNMP PDU

The seven SNMP protocol data units (PDUs) are as follows:

GetRequest A manager-to-agent request to retrieve the value of a variable or list of variables. Desired variables are specified in variable bindings (values are not used). Retrieval of the specified variable values is to be done as an atomic operation by the agent. A Response with current values is returned.

SetRequest A manager-to-agent request to change the value of a variable or list of variables. Variable bindings are specified in the body of the request. Changes to all specified variables are to be made as an atomic operation by the agent. A Response with (current) new values for the variables is returned.

GetNextRequest A manager-to-agent request to discover available variables and their values. Returns a Response with variable binding for the lexicographically next variable in the MIB. The entire MIB of an agent can be walked by iterative application of GetNextRequest starting at OID 0. Rows of a table can be read by specifying column OIDs in the variable bindings of the request.

GetBulkRequest Optimized version of GetNextRequest. A manager-to-agent request for multiple iterations of GetNextRequest. Returns a Response with multiple variable bindings walked from the variable binding or bindings in the request. PDU specific non-repeaters and max-repetitions fields are used to control response behavior. GetBulkRequest was introduced in SNMPv2.

Response Returns variable bindings and acknowledgement from agent to manager for GetRequest,SetRequest, GetNextRequest, GetBulkRequest and InformRequest. Error reporting is provided by error-status and error-index fields. Although it was used as a response to both gets and sets, this PDU was called GetResponse in SNMPv1.

Trap Asynchronous notification from agent to manager. Includes current sysUpTime value, an OID identifying the type of trap and optional variable bindings. Destination addressing for traps is determined in an application specific manner typically through trap configuration variables in the MIB. The format of the trap message was changed in SNMPv2 and the PDU was renamed SNMPv2-Trap.

InformRequest Acknowledged asynchronous notification from manager to manager. This PDU use the same format as the SNMPv2 version of Trap. Manager-to-manager notifications were already possible in SNMPv1 (using a Trap), but as SNMP commonly runs over UDP where delivery is not assured and dropped packets are not reported, delivery of a Trap was not guaranteed. InformRequest fixes this by sending back an acknowledgement on receipt. Receiver replies with Response parroting all information in the InformRequest. This PDU was introduced in SNMPv2.

2.3.4 SnmpV3

SNMPv3 is defined by RFC 3411-RFC 3418 (also known as 'STD0062'). SNMPv3 primarily added security and remote configuration enhancements to SNMP.

SNMPv3 provides important security features:

- Confidentiality - Encryption of packets to prevent snooping by an unauthorized source.
- Integrity - Message integrity to ensure that a packet has not been tampered within transit.

- Authentication - to verify that the message is from a valid source.

As of 2004 the IETF recognizes Simple Network Management Protocol version 3 as defined by RFC 3411-RFC 3418 (also known as STD0062) as the current standard version of SNMP. The IETF has designated SNMPv3 a full Internet standard, the highest maturity level for an RFC.

3

Problem Definition

Simulation of events to verify EMS functionality (For Fault management module) and improve and optimize FM (Fault Management)module in a way that performance and quality of FM module increased.

Scope

As the title suggests the goal of this project, work carried out in this research is useful in improving performance and quality of the Element Management system.

This project is divided into two goal, one improve the quality of the EMS by removing the bottleneck through simulation of Events of FM module and second, enhance the performance of FM module by applying multi-threading concept and solving synchronization problem in SNMP protocol.

Part 1 : Improve Quality by removing bottleneck through simulation of Events, because at the end of simulating events give a report which contains whether expected result has been got for all events and it also gives time taken by all events. If result is not same as expected result then do changes in source code and improve quality.

3.1 Events

Description of main events

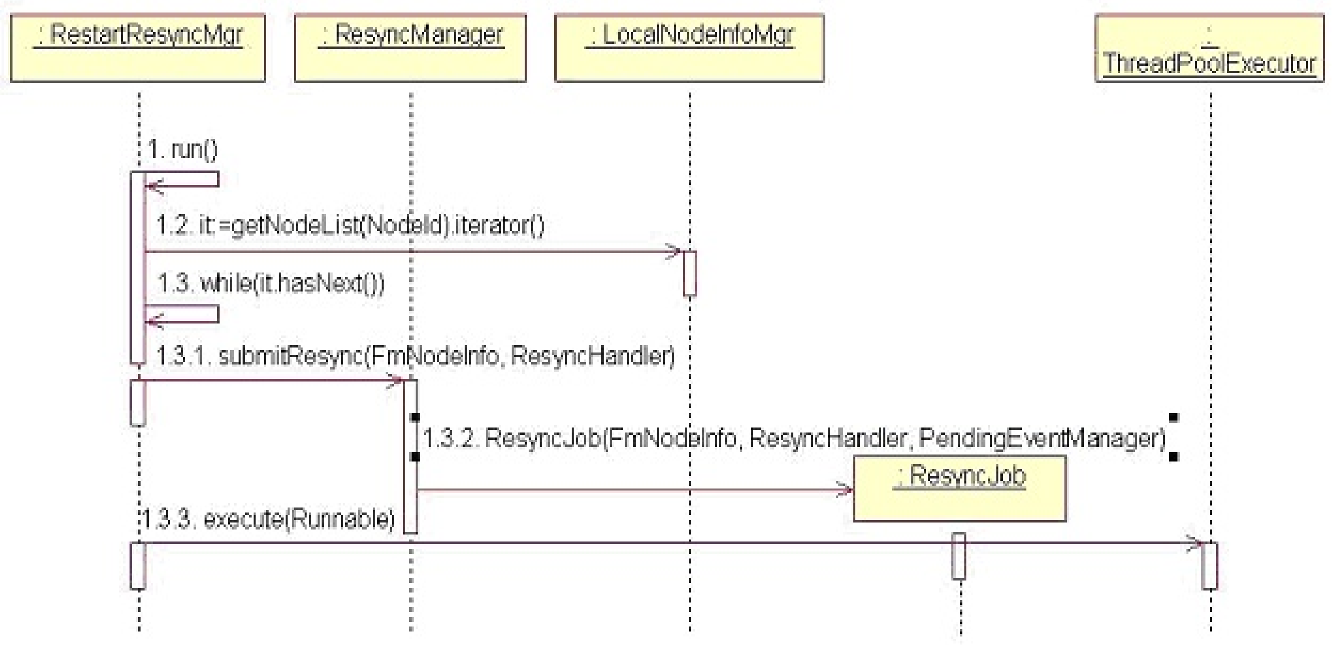
3.1.1 Resync Event

FM Resync Processing : A Resync Mechanism that enables the EMS to acquire from NE its complete snapshot of current state profile and active alarms list by sftp file, besides the EMS's normal NE event processing.

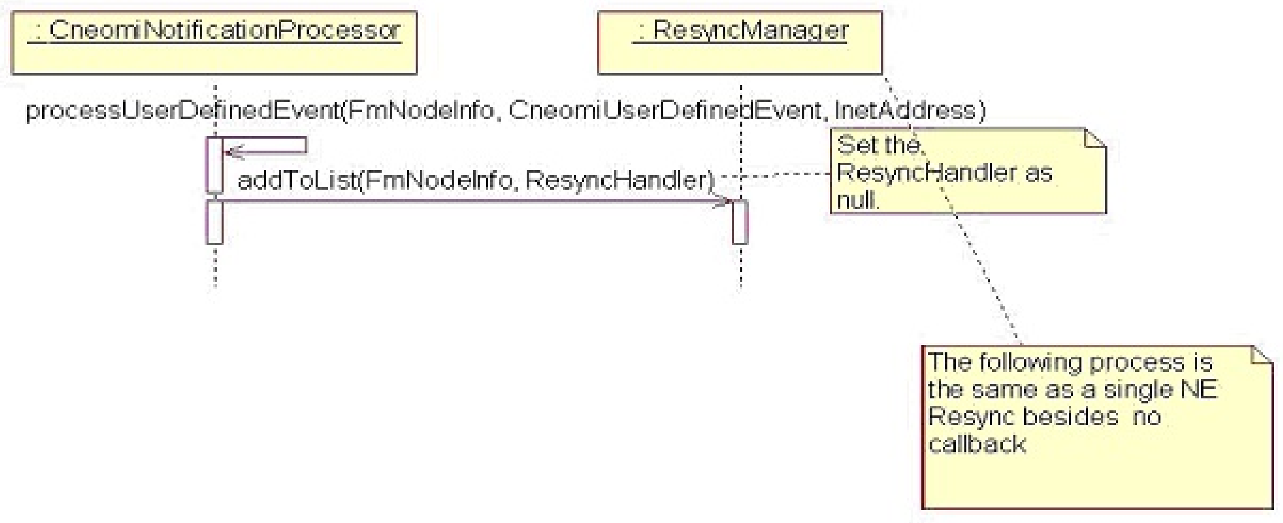
What this feature is:

- An entire resync operation for all resyncable NEs(CAPC and AP with their MO FRUs) when EMS startup
- The resync operation against one resyncable NE (CAPC and AP) after its reboot
- The resync operation for specified NE manually triggered by operator from Node Properties GUI.
- The reinforcement for the EMS's normal NE event processing, for which the resync processing must provides interface to maintain the proper event processing order per NE.

EMS startup Resync scheduling


: RestartResyncMgr
: ResyncManager
: LocalNodeInfoMgr
: ThreadPoolExecutor
1. run()
1.2. it:=getNodeList(NodeId).iterator()
1.3. while(it.hasNext())
1.3.1. submitResync(FmNodeInfo, ResyncHandler)
1.3.2. ResyncJob(FmNodeInfo, ResyncHandler, PendingEventManager)
: ResyncJob
1.3.3. execute(Runnable)

NE Reboot Resync scheduling


: CneomiNotificationProcessor
: ResyncManager
processUserDefinedEvent(FmNodeInfo, CneomiUserDefinedEvent, InetAddress)
addToList(FmNodeInfo, ResyncHandler)
Set the ResyncHandler as null.
The following process is the same as a single NE Resync besides no callback

3.1.2 Statechange Event

NE(AP/CAPC) can be in different state like UP, DOWN ,IMPAIRED, NOT PRESENT etc. State of the NE is decided based on different Action.

BootStrapAction : When new NE will be added, NE has to upload XML file which contains details like SNMP keys, object Id etc.. So EMS can get detail of each NE and can store in database for future use.

Sample of bootstrap XML file:

```
<?xml version="1.0" encoding="UTF-8"?> <SecurityBootstrapData
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\WiMax\EMS-NE ICD\test1.xsd">
<Diffie-HellmanParameters>
<Prime>"FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E0
88A67CC74020BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A
6DF25F14374FE1356D6D51C245E485B576625E7EC6F44C42E9A63A3620FFFFF
FFFFFFFFFFF"
</Prime> </Diffie-HellmanParameters> <snmpTargetAddrTable>
<snmpTargetAddrEntry    snmpTargetAddrName="800000A103111111111
111">
<snmpTargetAddrTAddress>203.8.31.30</snmpTargetAddrTAddress>
</snmpTargetAddrEntry> </snmpTargetAddrTable> <usmUserTable>
<usmUserEntry usmUserEngineID="800000A103222222222222"
usmUserName="admin_1">
<usmUserSecurityName>admin_1</usmUserSecurityName>
<usmUserAuthProtocol>usmHMACSHAAuthProtocol</usmUserAuthProtocol>
<usmUserPrivProtocol>usmAESPrivProtocol</usmUserPrivProtocol>
</usmUserEntry> <usmUserEntry
usmUserEngineID="800000A103222222222222" usmUserName="operator">
<usmUserSecurityName>operator</usmUserSecurityName>
<usmUserAuthProtocol>usmHMACSHAAuthProtocol</usmUserAuthProtocol>
<usmUserPrivProtocol>usmAESPrivProtocol</usmUserPrivProtocol>
</usmUserEntry> <usmUserEntry
usmUserEngineID="800000A103222222222222" usmUserName="guest">
<usmUserSecurityName>guest</usmUserSecurityName>
<usmUserAuthProtocol>usmHMACSHAAuthProtocol</usmUserAuthProtocol>
<usmUserPrivProtocol>usmAESPrivProtocol</usmUserPrivProtocol>
```

```
</usmUserEntry> </usmUserTable>
```

StateChangeEvent Action : State of NE can be changed explicitly through this action.

AlarmAction : When something goes wrong in NE, NE generates alarm and send it to EMS. EMS change the state of NE based on alarm. Ex if state of NE is UP and when EMS gets an alarm from NE, EMS will change the state of NE from UP to IMPAIRED.

NE Added Removed Action : This action will be generated when new NE will be added or removed.

Fru Added Removed Action : This action will be generated when new Fru will be added or removed.

HeartbeatAction : EMS generates this action at regular interval of time to check the status of NE whether NE is UP or DOWN. If EMS will not get response from NE before predefined time, it will make NE state down.

NE overall state Transition diagram

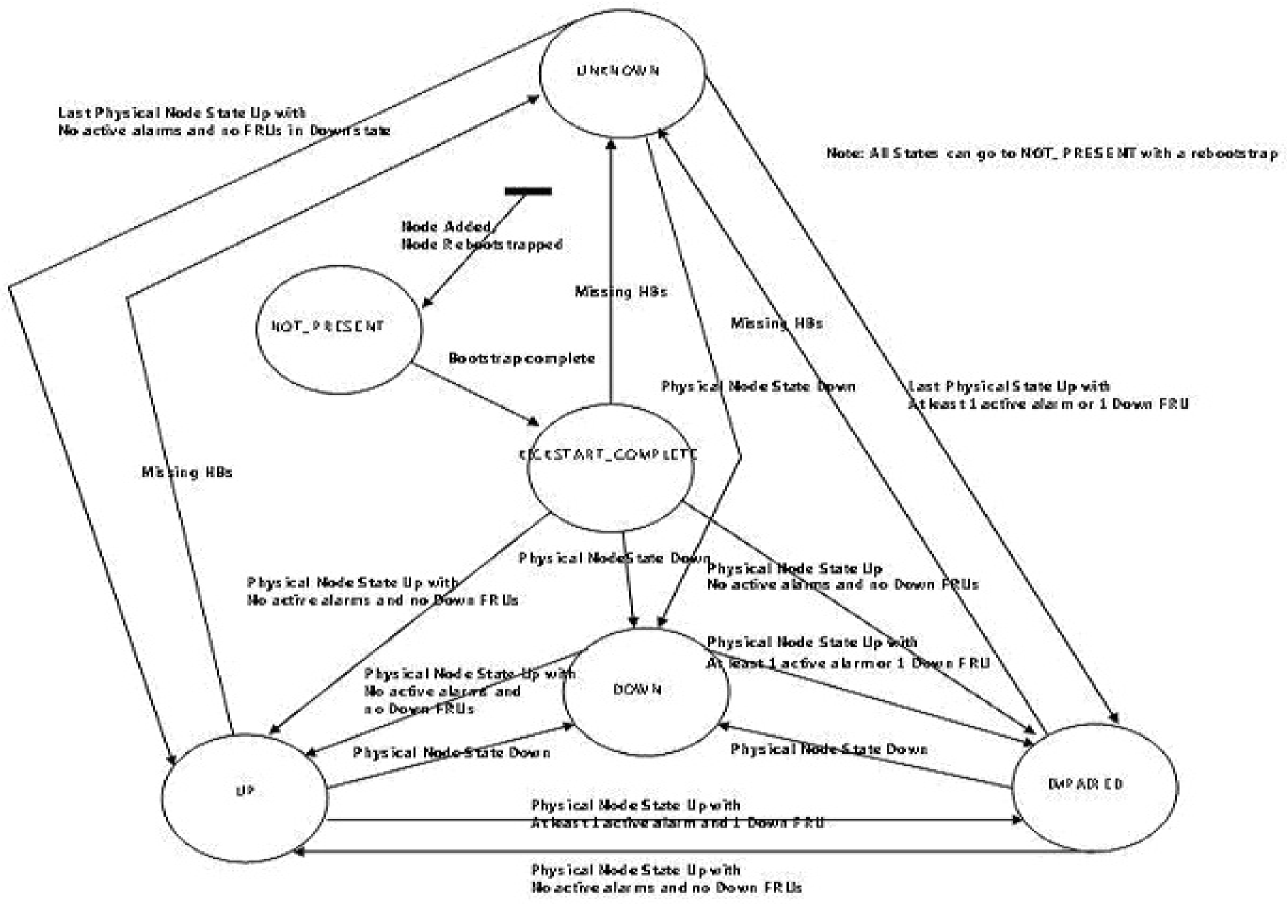

UNKNOWN
Last Physical Node State Up with
No active alaims and so FRUs in Down state
Note: All States can go to NOT_PRESENT with a rebootstrap
Node Added,
Node Rebootstrapped
NOT_PRESENT
Missing HBs
Missing HBs
Bootstrap complete
Physical Node State Down
Last Physical State Up with
At least 1 active alarm or 1 Down FRU
KICKSTART_COMPLETE
Missing HBs
Physical Node State Down
Physical Node State Up
No active alarms and no Down FRUs
Physical Node State Up with
No active alarms and no Down FRUs
Physical Node State Up with
At least 1 active alarm or 1 Down FRU
Physical Node State Up with
No active alarms and
no Down FRUs
DOWN
Physical Node State Down
Physical Node State Down
UP
IMPAIRED
Physical Node State Up with
At least 1 active alarm and 1 Down FRU
Physical Node State Up with
No active alarms and no Down FRUs

Figure 3.1: NE overall state transition diagram

State Inputs

Each state input shall at a minimum include the NE's NodeId as the mechanism to identify the associated NE node. The list below contains all possible inputs that trigger an overall NE node state calculation and node state modifications. These inputs shall implement an interface that will enable the State Mechanism to process a variety of different input types.

1. Node Connectivity Change - Connection Loss and Connection Reestablished
2. Alarm Set
3. Alarm Cleared
4. StateChange Event

5. Bootstrap Modification - Bootstrap Started and Bootstrap Completed
6. Resync (Information to completely refresh the state store associated with an NE that includes the presence of active alarms)

Node Connectivity Change Connection Loss and Connection Reestablished Change The Node Connectivity Change input is received when the EMS loses connectivity to the NE and when the EMS regains connectivity to the NE. For a connection loss, this input is reported by the HeartBeatMonitor thread. After receiving this input, the State Mechanism shall transition the NE's overall node state to UNKNOWN and rolldown this state to its displayable FRUs (Refer to UNKNOWN State Rolldown.) This transition will only occur if the NE's current state has not already transitioned to UNKNOWN or is not DOWN, (from not UNKNOWN to UNKNOWN) (from UNKNOWN to not UNKNOWN) and if the NE has been bootstrapped. For a connection reestablished, this input is generally reported by the HeartBeat thread. It shall trigger the transition from the UNKNOWN state to the displayable node's previous state (Refer to UNKNOWN State Rolldown).This input contains no additional data other than the NE's Node ID. If modified, the current displayable node statuses shall be made available for distribution. Note: displayable node statuses include the current NE overall node status and the displayable FRUs.

Alarm Set The Alarm Set input is received when the EMS receives an active alarm from the NE. It is reported by the HeavyWeight thread that processes alarms and statechange events. It is generated to trigger a potential transition of the NE's overall node state to IMPAIRED. The NE's overall node state can only transition to this state if it is bootstrapped, connected to the EMS, and its current state is not DOWN. Only type and Node Id are necessary for this input. It is a light weight input since no database accesses is required. This input only impacts the NE node's overall state. There is no rolldown of the IMPAIRED state to the NE's displayable FRU nodes. This input is ignored if the current NE overall node state is already IMPAIRED or UNKOWN. If modified, the NE overall node status update shall be made available for distribution.

Alarm Cleared The Alarm Clear input is received when the EMS receives a clear alarm from the NE. It is reported by the HeavyWeight thread that processes alarms and statechange events. It is generated to trigger a possible recalculation of the NE's overall node state. If the NE's overall node state is in the IMPAIRED state and there are no active FRUs, then the database should be queried for active alarms. If there are no active alarms, then the NE's overall node state shall transition to the UP state. Only the Node Id is necessary for this input. It is can be a heavy weight input since database access may occur. This input only impacts the NE node's overall state. There is no rolldown UP state. This input is ignored if the NE's overall node current state is not IMPAIRED or UNKNOWN. If modified, the current NE overall node status shall be made available for distribution.

StateChange Event The StateChange Event input is received when the EMS receives a statechange event from the NE. It is reported by the HeavyWeight thread that processes alarms and statechange events. This input may trigger a recalculation of the NE's overall node state. `Displayable FRU StateChange Event:` (Shannon) we shall do something for Displayable FRU status. When a current NE overall node state of UP/IMPAIRED/DOWN If an input for a statechange event for a displayable FRU is received with an operational state of Down and it is not already down, the current state for that displayable FRU shall transition to the DOWN state and its previous states shall be set to null. The current NE's overall node state shall transition to the IMPAIRED state only if it was in the UP state. If modified, the current displayable node statuses shall be made available for distribution.

When the current NE overall node state is UP/IMPAIRED/DOWN If an input for a statechange event for a displayable FRU is received with an operational state of something other than DOWN and it is not already in that state, then the current state of that displayable FRU shall transition to that state and its pre- vious state shall be set to null. The current NE's overall state shall be reevaluated if it is not already in the UP state. (See NE overall state calculation). If modified, the current displayable node statuses shall be made available for distribution.

When the current NE overall node state is UNKNOWN In this condition, if an input for a statechange event for a displayable FRU is received with an operational state that is different from its previous state, then its previous state shall transition to that state and its current state shall remain UN-KNOWN. No NE overall state calculation shall occur and no displayable status shall be distributed. (Shannon) I did not see the reason for state change that way. The event reception does not mean a connection establishment? This part shall also be read with reference to UNKNOWN State Rolldown.

Note: It is necessary to process state information when the NE is in the UN-KNOWN state to enable the rollback to a previous state when the NE connection is reestablished. This rollback to a last known state is a requirement.

Bootstrap Modification Bootstrap Started and Bootstrap Completed There are two types of input related to bootstrap modfication. One for a bootstrap request which is generated when a new managed NE is added or following an operator initiated rebootstrap operation. The other is for a bootstrap completion which occurs when the NE and the EMS have exchange keys and are capable of communicating via SNMP v3 protocol.

Bootstrap Request This type of input will transition the current NE's overall node state to NOT PRESENT along with all of its displayable FRUs node states if they are not already in that state. The previous states of the NE's displayable nodes shall be erased and the container of non-displayable FRU states cleared. Finally, the current displayable node statuses shall be made available for distribution.

Bootstrap Completion This type of input will transition the current NE's overall node state to KICK-START COMPLETE along with its displayable FRU nodes if the current NE's overall state is in the NOT PRESENT state. The previous states of the NE's dis-playable nodes should already be set to null. Finally, the current displayable node statuses shall be made available for distribution.

Resync Information to completely refresh the state store associated with an NE that includes the presence of active alarms The resync input is sent by the NE's ResyncJob after the resync process has completed processing its files and before it has processed its PendingQueue of alarms and statechange events. This input shall contain the following: a flag indicating if active alarms are present; the NE's Physical Node state; a list of dis-playable FRU states; and the list of DOWN non-displayable FRUS. The FRU state information shall include the FRU's identity (Managed Object Class and Managed Object ID) and its operational state.

When the current NE overall node state is in UNKNOWN In this scenario, the NE's non-displayable FRU states shall be refreshed, the NE's Physical Node State shall be updated, and the previous NE's displayable FRU states are updated. No NE overall node state calculation is performed and no current displayable statuses are distributed.

When the current NE overall node state is DOWN/KICKSTART COMPLETE/UP/ IMPAIRED In this scenario, the NE's non-displayable FRU states shall be refreshed; the NE's Physical Node State shall be updated along with the current NE's displayable FRU states. The current NE overall node state shall be calculated. All previous node states shall be erased. The current displayable node statuses shall be made available for distribution.

Note: if the NE's overall node state is NOT PRESENT, disregard this resync input.

Part 2 : Enhance performance of FM module by applying multithreading concept and solving synchronization problem in SNMP protocol.

3.2 Performance Issues

There are three performance issues

1. Single threading concept in Resync
2. Single threading concept in processing events
3. Out of synchronization between Manager and Agent, which are communicating using SNMP Protocol

3.3 Detail description of Each Issue

3.3.1 Single threading concept in Resync

- EMS must have latest information regarding state and alarm of all NEs, there is no issue when EMS is up. It will gather all information regarding the same.
- But when EMS goes down and after some time when it comes up, on that time to get updated information from all NEs, it sends request to all NEs.
- All NEs response back with latest information by uploading state and alarm file.
- But here while sending request to all NEs (more than 1000 NEs) EMS is using only single thread to fulfill this task and because of that it consumes too much time to complete whole process.
- And because of that performance of EMS is going down.

3.3.2 Single threading concept in processing Events

- Whenever problem occurs in NE side, It sends an Alarm to EMS. On getting Alarm EMS does process on that and basis on the Alarm or Event, it will change the state of NE.

  Impaired state : if it receives an alarm (yellow color)

  Down state : if it receives an down state change event (NE is not working, Red color) etc

Event Types :

1. Heartbeat Event
2. Alarm
3. Cneomi management Event

Problem:

- There are more than 1000 NEs in the network, so many alarms or events come within the sort interval of time.

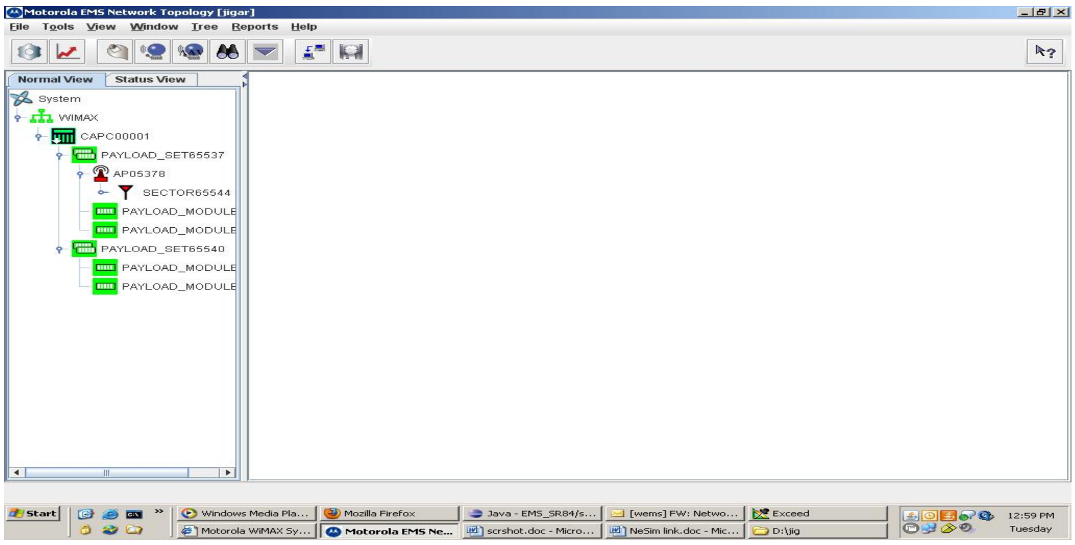


Figure 3.2: Down NEs are in red color

- EMS has to process all the events or alarms to update the status of each and every NE.

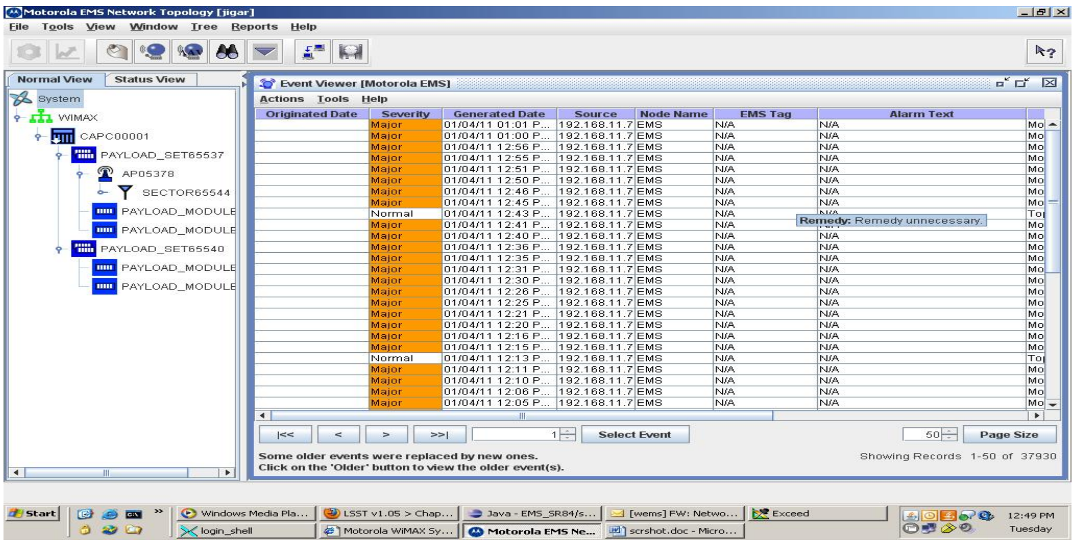
How EMS is handling events or alarms.

- Obviously some parallel process is required to process this many alarms within sort interval.
- EMS used to handle with multithreading concept. For each type of event EMS was using different thread. For heartbeat -¿ heartbeat thread For cneomi management event -¿ light weight thread For alarm -¿ heavy weight thread But, here only three threads were running each having their on queue of event data which it operates on. Because of only three threads it was consuming too much time to process all events.

3.3.3 Out of synchronization between Manager and Agent, which are communicating using SNMP Protocol

Optimization In Communication

- Snmp take cares of two types of classes
- Confirmed Class : The Confirmed Class contains all protocol operations which cause the receiving SNMP engine to send back a response. For example, [RFC3416] defines the following operations for the Confirmed Class: GetRequest-PDU, GetNextRequest-PDU,



Originated Date	Severity	Generated Date	Source	Node Name	EMS Tag	Alarm Text	
	Major	01/04/11 01:01 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 01:00 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:56 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:55 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:51 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:50 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:46 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:45 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Normal	01/04/11 12:43 P...	192.168.11.7	EMS	N/A	N/A	To
	Major	01/04/11 12:41 P...	192.168.11.7	EMS	N/A	[illegible]	Mo
	Major	01/04/11 12:40 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:36 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:35 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:31 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:30 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:26 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:25 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:21 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:20 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:16 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:15 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Normal	01/04/11 12:13 P...	192.168.11.7	EMS	N/A	N/A	To
	Major	01/04/11 12:11 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:10 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:06 P...	192.168.11.7	EMS	N/A	N/A	Mo
	Major	01/04/11 12:05 P...	192.168.11.7	EMS	N/A	N/A	Mo

Remedy: Remedy unnecessary.

Some older events were replaced by new ones.
Click on the 'Older' button to view the older event(s).

Showing Records 1-50 of 37930

Figure 3.3: Event viewer shows all alarms and events

GetBulkRequest-PDU, SetRequest-PDU, and InformRequest-PDU. Note : In this type of class authenticator is Receiver

- Unconfirmed Class : The Unconfirmed Class contains all protocol operations which are not acknowledged. For example, [RFC3416] defines the following operations for the Unconfirmed Class: Report-PDU, Trapv2-PDU, and GetResponse-PDU. Note: In this type of class authenticator is Sender Authenticator means a lot here, because packet has Engine Boot value and Engine Time value of authenticator, which require to prevent from REPLAY attack. You can find more detail in next section.

Replay Protection Each SNMP engine maintains three objects:

- snmpEngineID, which (at least within an administrative domain) uniquely and unambiguously identifies an SNMP engine.

- snmpEngineBoots, which is a count of the number of times the SNMP engine has re-booted/re-initialized since snmpEngineID was last configured; and,

- snmpEngineTime, which is the number of seconds since the snmpEngineBoots counter was last incremented.

Each SNMP engine is always authoritative with respect to these objects in its own SNMP entity. It is the responsibility of a non- authoritative SNMP engine to synchronize with the authoritative SNMP engine, as appropriate.

An authoritative SNMP engine is required to maintain the values of its snmpEngineID and snmpEngineBoots in non-volatile storage.

msgAuthoritativeEngineID

The msgAuthoritativeEngineID value contained in an authenticated message is used to defeat attacks in which messages from one SNMP engine to another SNMP engine are replayed to a different SNMP engine. It represents the snmpEngineID at the authorita- tive SNMP engine involved in the exchange of the message.

When an authoritative SNMP engine is first installed, it sets its local value of snmp EngineID according to a enterprise-specific algorithm (see the definition of the Textual Convention for SnmpEngineID in the SNMP Architecture document [RFC3411]).

msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime

The msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime values contained in an authenticated message are used to defeat attacks in which messages are replayed when they are no longer valid. They represent the snmpEngineBoots and snmpEngineTime values at the authoritative SNMP engine involved in the exchange of the message.

Through use of snmpEngineBoots and snmpEngineTime, there is no requirement for an SNMP engine to have a non-volatile clock which ticks (i.e., increases with the passage of time) even when the SNMP engine is powered off. Rather, each time an SNMP engine re-boots, it retrieves, increments, and then stores snmpEngineBoots in non-volatile storage, and resets snmpEngineTime to zero.

When an SNMP engine is first installed, it sets its local values of snmpEngineBoots and snmpEngineTime to zero. If snmpEngineTime ever reaches its maximum value (2147483647), then snmpEngineBoots is incremented as if the SNMP engine has re-booted and snmpEngineTime is reset to zero and starts incrementing again.

Each time an authoritative SNMP engine re-boots, any SNMP engines holding that authoritative SNMP engine's values of snmpEngineBoots and snmpEngineTime need to re-synchronize prior to sending correctly authenticated messages to that authoritative SNMP engine. Note, however, that the procedures do provide for a notification to be accepted as authentic by a receiving SNMP engine, when sent by an authoritative SNMP engine which has re-booted since the receiving SNMP engine last (re-)synchronized.

If an authoritative SNMP engine is ever unable to determine its latest snmpEngineBoots value, then it must set its snmpEngineBoots value to 2147483647.

Whenever the local value of snmpEngineBoots has the value 2147483647 it latches at that value and an authenticated message always causes an notInTimeWindow authentication failure.

In order to reset an SNMP engine whose snmpEngineBoots value has reached the value 2147483647, manual intervention is required. The engine must be physically visited and re-configured, either with a new snmpEngineID value, or with new secret values for the authentication and privacy protocols of all users known to that SNMP engine. Note that even if an SNMP engine re-boots once a second that it would still take approximately 68 years before the max value of 2147483647 would be reached.

Time Window The Time Window is a value that specifies the window of time in which a message generated on behalf of any user is valid. This memo specifies that the same value of the Time Window, 150 seconds, is used for all users.

Time Synchronization

Time synchronization, required by a non-authoritative SNMP engine in order to proceed with authentic communications, has occurred when the non-authoritative SNMP engine has obtained a local notion of the authoritative SNMP engine's values of snmpEngineBoots and snmpEngineTime from the authoritative SNMP engine. These values must be (and remain) within the authoritative SNMP engine's Time Window. So the local notion of the authoritative SNMP engine's values must be kept loosely synchronized with the values stored at the authoritative SNMP engine. In addition to keeping a local copy of snmpEngineBoots and snmpEngineTime from the authoritative SNMP engine, a non-authoritative SNMP engine must also keep one local variable, latestReceivedEngineTime. This value records the highest value of snmpEngineTime that was received by the non-authoritative SNMP engine from the authoritative SNMP engine and is used to eliminate the possibility of replaying messages that would prevent the non-authoritative SNMP engine's notion of the snmpEngineTime from advancing.

A non-authoritative SNMP engine must keep local notions of these values (snmpEngineBoots, snmpEngineTime and latestReceivedEngineTime) for each authoritative SNMP engine with which it wishes to communicate. Since each authoritative SNMP engine is uniquely and unambiguously identified by its value of snmpEngineID, the non-authoritative SNMP engine may use this value as a key in order to cache its local notions of these values.

Time synchronization occurs as part of the procedures of receiving an SNMP message. As such, no explicit time synchronization procedure is required by a non-authoritative SNMP engine. Note, that whenever the local value of snmpEngineID is changed (e.g., through discovery) or when secure communications are first established with an authoritative SNMP engine, the local values of snmpEngineBoots and latestReceivedEngineTime should be set to zero. This will cause the time synchronization to occur when the next authentic message is received.

Problem

During implementation / test of my snmp management application I ran into some issues with regards to timeliness management.

1. snmpEngineTime calculations uses "System.currentTimeMillis()" as reference. "System.currentTimeMillis()" is based on system time which may change forward / backwards at any time (like user setting of time / ntp activation / daylight saving / ....)

Simple testsequences (On Manager side)

- snmp.get
- modify systemtime
- snmp.get

- The effect of setting system time backwards in step 2 is that the snmpEngineTime used in messages will be increased with the time that systemtime was set backwards. This results in a usmStatsNotInTimeWindows report from the agent and the manager now recovers fine from the usage of the wrong snmpEngineTime
- The effect of setting system time forward in step 2 is that the snmpEngineTime used in messages will be decreased with the time that systemtime was set forward. This results in a usmStatsNotInTimeWindows report from the agent but in this case the manager "ignores" the report from the agent and tries to retransmit the message, thus ending in a timeout on the api level.

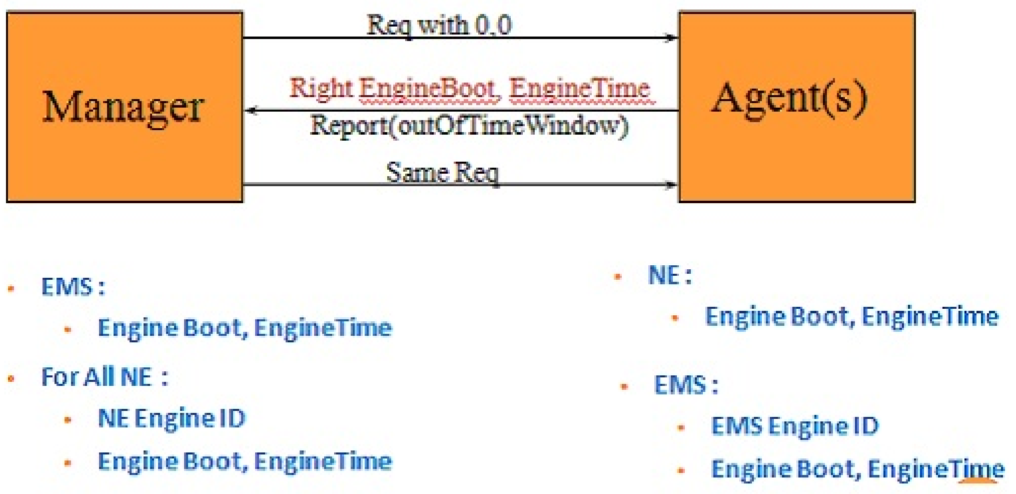
All future communication to the snmp agent will now fail with timeouts.

2. Changing system may just as well happen on the agent side as well, so I decided to make similar test for such cases. The agent I did such test on, is based on NESIM. It turned out that this agent implementation also is affected by changes to system time. (the manager application is still based on snmp4j as above)

The testsequence is similar to before:

- snmp.get
- modify time in agent
- snmp.get

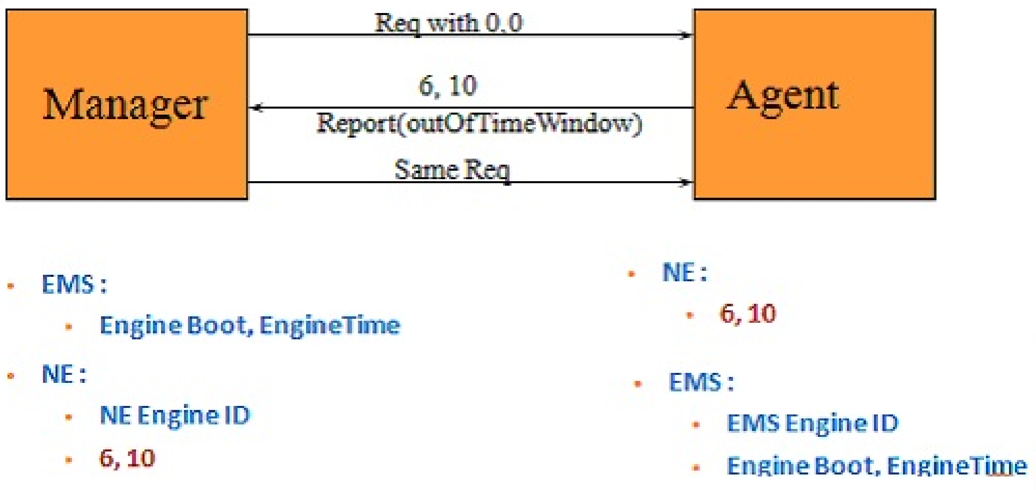
- Agent time is adjusted forwards with some value larger than 150 seconds. A usmStatsNotInTimeWindows-report is send from the agent to the manager in step 3 and the manager now recovers fine.
- Agent time is adjusted backwards with some value larger than 150 seconds. Behaviour is similar to 1b and end result is also that all future communication fails with timeout.



Example :

- First will see How communication start

Suppose EngineBoot and EngineTime of NE is 6 and 10.

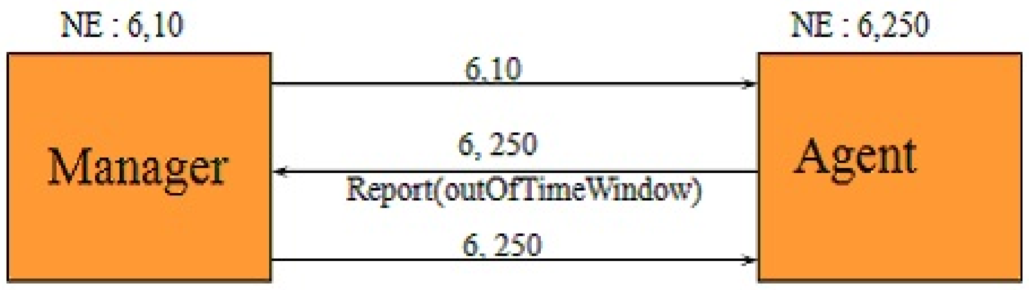


- Now four scenarios of simulation

1. Decrease EMS system time
2. Increase EMS system time
3. Decrease NE system time
4. Increase NE system time

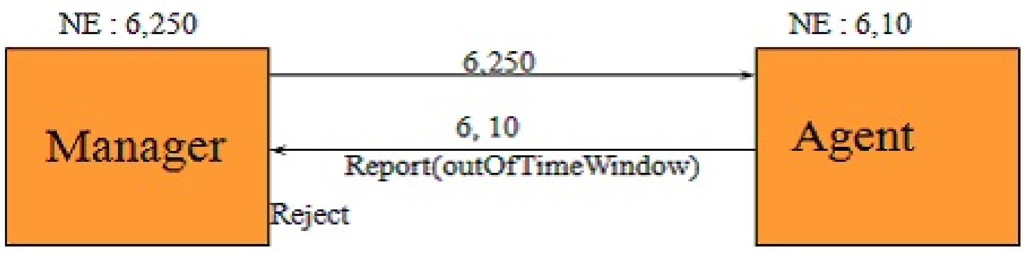
All four scenarios won't create problem but two.

1. Decrease EMS system time (It won't create problem)



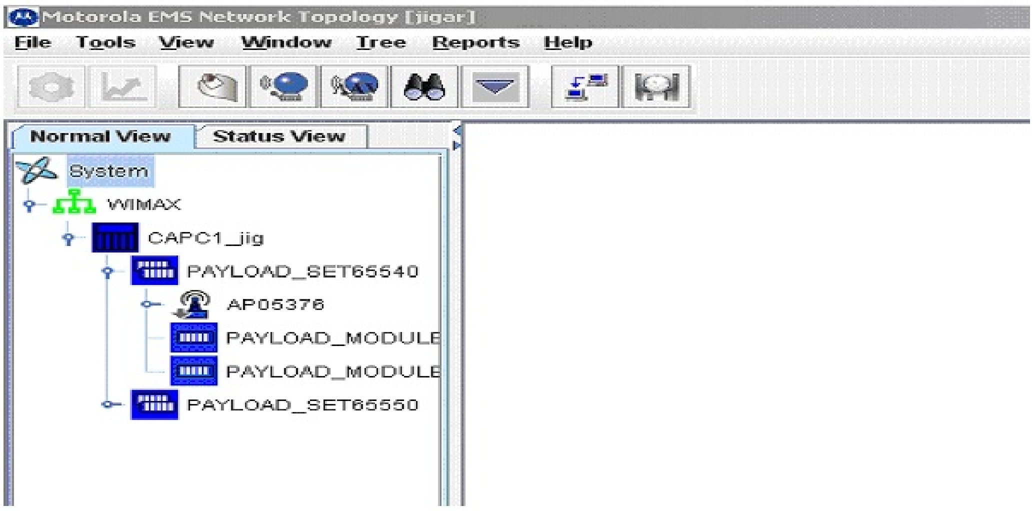
- On getting Report(outOfTimeWindow) EMS will update NE EngineBoot and EngineTime with correct value which it has got from NE side.

2. Increase EMS system time



- On getting Report(outOfTimeWindow) EMS will Reject this packet because Its an old Report.
- Problem lies here

- Now EMS continues to reject packet for the same reason.
- And because of that Heartbeat of all NEs also be rejected, which causes all NEs to go down in UNKNOWN state.



captionAll NEs are in Unknown state

4

Technology, Solution and Methodology

4.1 Tools and Technology

- Core Java
- SNMP Protocol (For Communication)
- Clearcase
- Emma Tool to measure Coverage
- NESIM (Simulator for WiMAX network)
- Eclipse

4.2 Solution

Here is a solution for all performance issues

4.2.1 Solution for Single threading concept in Resync

- Multithreading
- Problem with multithread (how to correlate Response with Request)
- Solution for that
  - Add Correlation Tag with Handler in HashMap
  - Correlation Tag is nothing but a counter which is used to correlate response with request

- On Response(of upload Request) fetch handler as per the correlation tag and do further processing

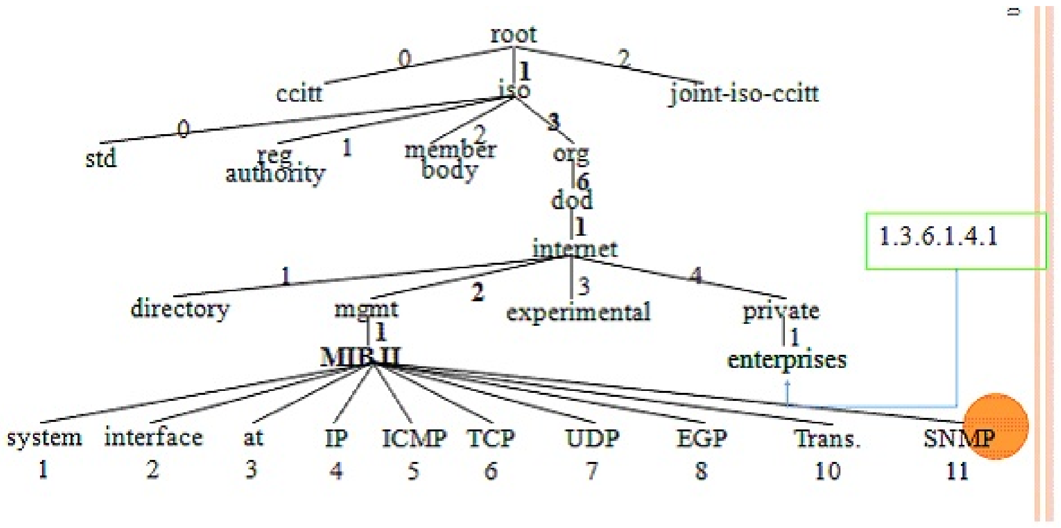
How SNMP comes into picture in this solution

Network Management Protocol

- Communication protocol between managers and agents
- Protocol provides a standard way to exchange management information between managers and agents

Object Identifier (OID):

- Global identifier for a particular object type.
- An OID consists of a sequence of integers, which specify the position of the object in the global object identifier tree.
- Ex : PDU contains OID, Value (OID for filepath, Path value)



captionHow to identify OID

Private MIB Registration

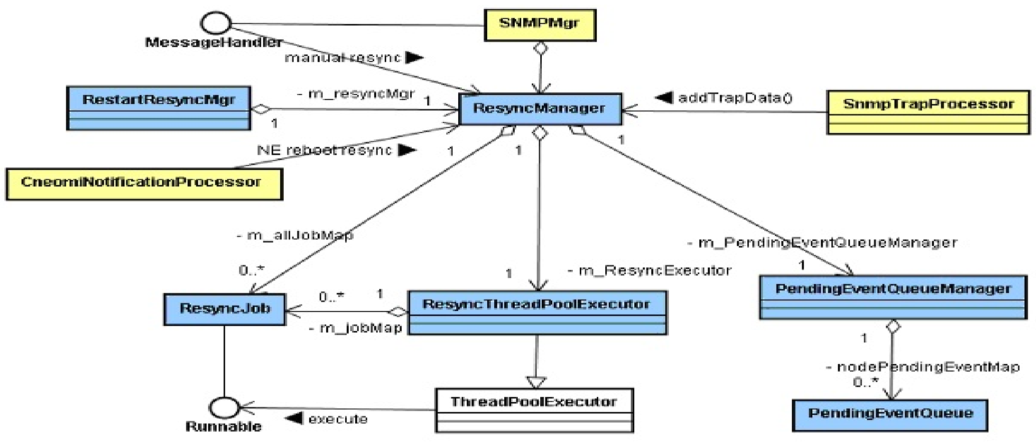
- Companies can register their private MIB extensions in the global MIB tree by contacting the Internet Assigned Numbers Authority (IANA).
  - http://www.iana.org/

- Company broadcast their ID
- Ex Motorola : 1.3.6.1.2.2225

Interface

- To achieve all this layer on top of Snmp needs to be developed where all extra OIDs can be added.
- And that OIDs can be set in PDU as a variable binding with its value as per the Snmp Protocol.
- This OIDs are hard coded on both the side which communicate with each other so both the party can identify the field.

Class Diagram



Class Specification SNMPMgr:

- Implementing MessageHandler, handling 'SNMP RESYNC ALARM STATE REQUEST' message, scheduling resync for specified NE by operator manually from Node Properties GUI.

RestartResyncMgr:

- Up to an entire resync operation for all resyncable NEs(CAPC and AP with their MO FRUs) when EMS startup

CneomiNotificationProcessor:

- Handling inventory resync event, scheduling resync operation against one rebooted NE

ResyncManagers:

- As the ResyncJob pool, to cache and execute all the resync operations

ResyncThreadPoolExecutor:

- Extending ThreadPoolExecutor, as the ResyncManager kernel to schedule and execute the ResyncJobs

ResyncJob:

- Resync procedure for one NE

PendingEventQueue:

- A queue to cache the event from SNMPTrapProcessor for future processing by ResyncJob during resync process, one queue for one NE, one ResyncJob, unique with NodeId.

PendingEventQueueManager:

- A managing class for all PendingEventQueue

4.2.2 Solution for Single threading concept in processing events

- Here heartbeat event comes after every 40 seconds to say NE is alive. Frequency of cneomi management event is also low such that it can be handled by one thread. But frequency of alarm is too high.
- So obvious solution is to increase the number of heavyweight threads which handles all alarm such a way that performance of EMS will be enhanced by consuming less time to process alarm.

  For heartbeat $\rightarrow$ 1 heartbeat thread

  For cneomi management event $\rightarrow$ 1light weight thread

  For alarm $\rightarrow$ 10 heavy weight threads

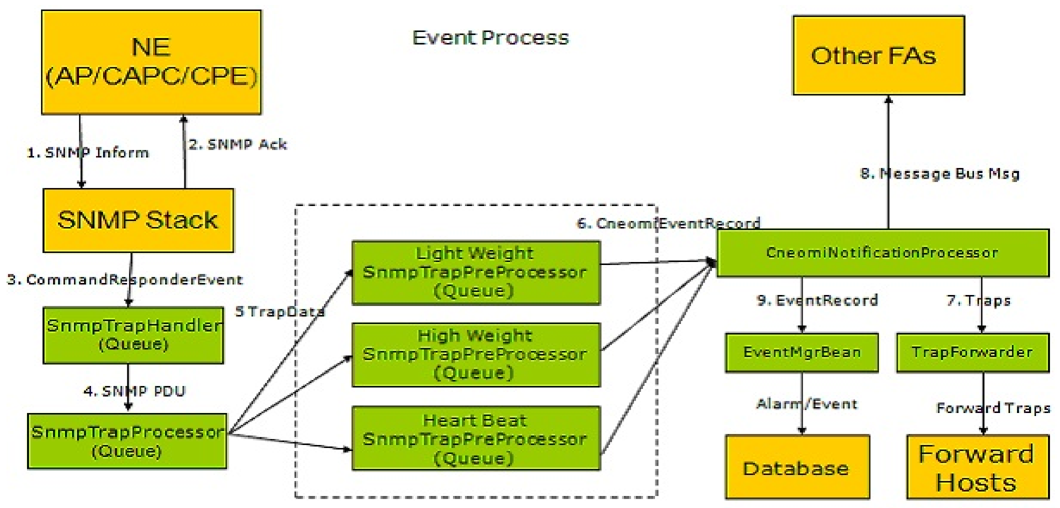
Class Diagram

- Each thread is having individual queue for event data.

  How heavy weight thread distribute each event data :

```
m_hw_thread_size = 10 int nodeId = fmNodeInfo.getNodeId().getInt();
m_hwList[(nodeId % m_hw_thread_size)].addToEventQ(event, fmNodeInfo
```

Flow:



4.2.3 Solution for Out of synchronization between Manager and Agent, which are communicating using SNMP Protocol

Root Cause of the problem :

- EMS and NE both are dependant on NTP server for Time.
- Setting the clock with NTP for our boxes that lack a battery backed clock. They start up in 1970 and suddenly (back to the future)they are in 2004. Looking at the code there also seemed to be an overflow problem occuring after 248 days and some tests seemed to verify this. Both large time changes and the 248 days overflow causes the agent to get stuck in "notInTimeWindow"-mode.
- It is all related to the fact that the engineTime is calculated with gettimeofday and in centiseconds. It is then stored in a signed (normally 32 bit) integer. On most platforms this value overflows after $2\hat{3}1100$ seconds 248,5 days.

Proposed Solution

(1)

- SNMP uses System.currentMilliSecond() to calculate time.Instead of that we have started using System.nenoTime which givs time in neno second, precision wise it gives more accurate result.
- Though this can not solve this problem, but this can delay overow by some more days. So problem can be delayed by some more days

(2)

- Objective : On getting Report from NE side saying usmStatsNotInTimeWindow ...call new ReportHandler and set Engine Boot and Engine Time for that particular NE as 0 , 0 on manager side. So communication won't stop between manager and agent. New ReportHandler has been written to handle usmStatsNotInTimeWindow.

- Scenario :

  - First Manager and Agent both are not in sync $\rightarrow$ On that time manager will send SNMP SET request to NE and NE will reply back with Report (usmStatsNotInTimeWindow) and manager will set that (Engine Boot and Engine Time).
  - Now communication is going smoothly as both r in sync. But if I change the time of NESIM (Decrease system time), so both wont be in sync. Now when manager sends SET or GET request to NE, NE will reply back with Report (usmStatsNotInTimeWindow), because manager has higher value of NE Boot or time difference is more than 150.

- On getting this Report, ReportHandler gets invoke from SNMP stack and set the Engine Boot and Engine Time for that particular NE as 0 , 0 on manager side. And from second request Manager will set Time and Boot value from NE as describe above. So communication won't stop between manager and agent.

5

Implementation and Result

Part 1, Implementation: Improve Quality by removing bottleneck through simulation of Events, because at the end of simulating events give a report which contains whether expected result has been got for all events and it also gives time taken by all events.

If result is not same as expected result then do changes in source code and improve quality.

Simulation of Event:

```
public void run() {
       FmNodeInfo fmNodeInfo = null;
       try{
           NodeId nodeId = nodeInfo.getNodeId();
      try {
                fmNodeInfo =(FmNodeInfo)LocalNodeInfoMgr.getIns
                tance().getNode(nodeId);
                if(fmNodeInfo == null)
                {
                  logger.warning("The nodeinfo for this nodeid: "
                  + nodeId.toString()
                  + " may be deleted or stale.");
                  return;
                }
              logger.finest("The new nodeinfo is: " + fmNodeInfo.
              toString());
              }
          catch (NMSException e) {
```

```
    logger.warning("There is a exception when query
    node by nodeId: " + e.getErrorCode()+
              "" + e.getMsg());
          return;
        }
    if( !ResyncUtil.isResyncable( fmNodeInfo ) )
    {
             logger.warning("The nodeinfo for this nodeid: " +
             nodeId.toString() + " is not resyncable.");
             return;
     }
     this.nodeInfo = fmNodeInfo;
     this.logger.info("ResyncJob begins for "+DescriptionBuild
     er.getNodeDisplayName(this.nodeInfo));
     this.resynPrg.getTimer().time();
      //begining
      //begin Pending Events
     pendingEventQueue=this.pendingEventQueueManager.beginPend
     ingEvent(this.getNodeInfo().getNodeId();
     sendUploadCommands();
     ResyncStatus status=this.resynPrg.awaitUploadComplete();
     this.parseData(status);//first parse according to the status
     status=this.resynPrg.awaitUploadComplete();
     this.parseData(status);//the second parse
     this.updateNEState();
     ResyncResult result=this.resynPrg.getResyncResult();
     this.executeResyncHandlers(result);
 }
catch(Throwable t) {
         this.logger.log(Level.SEVERE, "ResyncJob for "
         +Descri ptionBuilder.getNodeDisplayName(this.nodeInfo)
         +" breaks down.", t);
         this.resynPrg.updateProcStatus(ResyncStatus.FAIL,
         "aborted by exception:"+t.getMessage());
         ResyncResult result=new ResyncResult(ResyncResult.
         Status.FAILED,
         "aborted by exception:"+t.getMessage(), this.nodeInfo);
         this.executeResyncHandlers(result);
}finally{
```

```
          this.processPendingEvents();
//it's important to clear the pending event queue no matter the job
succeeds or fails. Or the pending event will be piled up.
this.resynPrg.getTimer().time();//the end
       ResyncResult result=this.resynPrg.getResyncResult();
       if(result==null)
              result=new ResyncResult(Status.ABORTED, "resync job
              aborted as node may be deleted", null,
              resynPrg.getTimer());
              this.executeResyncHandlers(result);
              this.logger.info(DescriptionBuilder.finalLogString
              (result));
      }
  }
```

Part 2, Implementation and Result: Enhance performance of FM module by applying multithreading concept and solving synchronization problem in SNMP protocol.

1. Multi threading concept in Resync

   Result :

Number of threads	Average Time (second)
5	42
9	30
10	30
11	31
15	40

   Implemented: 10 threads

2. Multi threading concept in processing events

```
  // Processor queues
    private SnmpTrapPreProcessor m_lwProcessor = null;
    private SnmpTrapPreProcessor m_hwProcessor = null;
    private SnmpTrapPreProcessor m_heartBeatProcessor = null;


    public static SnmpTrapPreProcessor m_hwList[] = null;
    public static int m_hw_thread_size = 10;

    public static final String FM_TRAP_HANDLER_QUEUE_SIZE =
    "FM_TRAP_HANDLER_QUEUE_SIZE";
    public static final int FM_TRAP_HANDLER_DEFAULT_QUEUE_SIZE =
    5000;
    private int m_queue_size = FM_TRAP_HANDLER_DEFAULT_QUEUE_SIZE;
    private BlockingQueue<CommandResponderEvent> msgQ = null;
    private String ipStr=null



      FmNodeInfo fmNodeInfo = FmNodeMap.getInstance()
      .getNodeFromCache
        (ipAddr.getInetAddress().getHostAddress());
      if (fmNodeInfo == null)
      {
        m_hwProcessor.addToEventQ (event, null,eventType);
        return;
      }

if (eventType != null) switch (eventType)
      {
        case COMMUNICATIONS_ALARM:
        case ENVIRONMENTAL_ALARM:
        case EQUIPMENT_ALARM:
        case PROCESSING_ERROR_ALARM:
        case QUALITY_OF_SERVICE_ALARM:
        case INTEGRITY_VIOLATION_EVENT:
        case OPERATIONAL_VIOLATION_EVENT:
        case PHYSICAL_VIOLATION_EVENT:
        case SEC_SERVICE_MECH_VIOLATION_ALARM:
        case TIME_DOMAIN_VIOLATION_EVENT:
        case CLMIB2_STATE_CHANGE_EVENT:
           /* if(ipAddr.toString().indexOf("/")>=0)
                  ipStr=ipAddr.toString().substring
                  (0,ipAddr.toString().indexOf("/"));
            int ipInt=new Integer(ipStr.charAt(ipStr.length()-1))
            .intValue();
            if(ipInt%2==0)
                m_hwProcessor.addToEventQ (event, fmNodeInfo);
            else
                m_hwProcessor_test.addToEventQ(event, fmNodeInfo);
                */
            /*if (ipAddr.toString().indexOf("/") >= 0) {
            ipStr =
                ipAddr.toString().substring(0,
                    ipAddr.toString().indexOf("/"));
        }*/
        int nodeId = fmNodeInfo.getNodeId().getInt();
        m_hwList[(nodeId % m_hw_thread_size)].addToEventQ(event,
        fmNodeInfo,eventType);
```

Result : It's depend on number and type of event are coming from NE side. But by applying multithreading cocept performance of processing event enhanced.

3. Out of synchronization between Manager and Agent, which are communicating using SNMP Protocol

```
public synchronized int checkTime(final UsmTimeEntry entry)
{
  int now = (int) (System.currentTimeMillis() / 1000);
  if (localTime.getEngineID().equals(entry.getEngineID()))
  {
    /* Entry found, we are authoritative */
    if ((localTime.getEngineBoots() == 2147483647) ||
        (localTime.getEngineBoots() != entry.getEngineBoots())
        ||
        (Math.abs(now + localTime.getTimeDiff() -
        entry.getLatestReceivedTime()) > 150)) {
      if (logger.isDebugEnabled()) {
        logger.debug(
            "CheckTime: received message outside time
            window (authorative):"+
            ((localTime.getEngineBoots() !=
              entry.getEngineBoots()) ? "engineBoots differ" :
```

```
            ""+(Math.abs(now + localTime.getTimeDiff() -
                         entry.getLatestReceivedTime()))+"
                         > 150"));
      }
      return SnmpConstants.SNMPv3_USM_NOT_IN_TIME_WINDOW;
    }
    else {
      if (logger.isDebugEnabled()) {
        logger.debug("CheckTime: time ok (authorative)");
      }
      return SnmpConstants.SNMPv3_USM_OK;
    }
  }
  else {
    UsmTimeEntry time = (UsmTimeEntry) table.get
    (entry.getEngineID());
    if (time == null) {
      return SnmpConstants.SNMPv3_USM_UNKNOWN_ENGINEID;
    }
    if ((entry.getEngineBoots() < time.getEngineBoots()) ||
        ((entry.getEngineBoots() == time.getEngineBoots()) &&
         (time.getTimeDiff() + now >
          entry.getLatestReceivedTime() + 150)) ||
        (time.getEngineBoots() == 2147483647)) {
      if (logger.isDebugEnabled()) {
        logger.debug(
            "CheckTime: received message outside time window
            (non authorative)");
      }
      return SnmpConstants.SNMPv3_USM_NOT_IN_TIME_WINDOW;
    }
    else {
      if ((entry.getEngineBoots() > time.getEngineBoots()) ||
          ((entry.getEngineBoots() == time.getEngineBoots())
          &&(entry.getLatestReceivedTime() > time
           .getLatestReceivedTime()))) {
        /* time ok, update values */
        time.setEngineBoots(entry.getEngineBoots());
        time.setLatestReceivedTime(entry
          .getLatestReceivedTime());
          time.setTimeDiff(entry.getLatestReceivedTime() - now);
        }
        if (logger.isDebugEnabled()) {
          logger.debug("CheckTime: time ok (non authorative)");
        }
        return SnmpConstants.SNMPv3_USM_OK;
      }
    }
  }
```

Result : SNMP stack will be reset for all NEs (with their EngineBoot and EngineTime) and communication will continue

6

Conclusion and Future work

6.1 Conclusion

Fault management module is a heart of WiMAX Element, so performance of WiMAX-EMS is enhanced by enhancing performance of FM.

Simulation of events has improved the quality by fixing loopholes in FM. Performance of FM module has been enhanced by applying multi-threading concept in processing Resync event and in processing alarm and performance has also been enhanced by resetting SNMP stack.

6.2 Future work

The current solution of synchronization problem in SNMP is just a workaround; there is nothing wrong in SNMP4j code. Root cause is in NTP server (server used for Time synchronization), so to solve this problem at server side will be the main priority.

Quality of WiMAX-EMS can be improved a lot by doing same thing for other module like configuration management etc.

7

References

1. RFC 3412
2. RFC 3414
3. compass.mot.com
4. `www.net-snmp.org/tutorial/tutorial-5/.../snmptrap.html`
5. `http://www.dpstele.com/layers/l2/snmp_l2_tut_part1.php`
6. `http://download.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/BlockingQueue.html`
7. EMS_Overview. Motorola Internal
8. FM_overview. Motorola Internal
9. NESIM_EMS_environment_setup. Motorola Internal
10. Wlan_default.property_and_Oracle_configuration. Motorola Internal
11. Bootstrap_overview_(SNMPV3). Motorola Internal
12. EMS_Active_Alarm_Viewer_LLD. Motorola Internal
13. FM State Handling Mechanism. Motorola Internal
14. FM_Resync_Process. Motorola Internal