Generic SNMP Proxy Agent Framework for Management of Heterogeneous Network Elements

Santosh S. Chavan[1] and R. Madanagopal[1]
[1]NMSWorks Software Pvt. Ltd, Chennai-600036, India
Phone: +91-44-22570433, Email: {santosh, madan}@nmsworks.co.in

Abstract— **Centralized management and monitoring of Network Elements (NEs) in a communication network is very critical to ensure high availability and lower downtime. Many management protocols like SNMP, CMIP, TL1, CORBA etc. have been standardized by different bodies to facilitate unified management of different types of NEs. However, many NEs do not support these standard management protocols for different reasons and instead provide proprietary management mechanisms. To facilitate management of these NEs, a proxy/mediation function is required. In this paper, we present a Generic Proxy Agent framework for management of heterogeneous Network Elements. This framework consists of two components, namely an SNMP Proxy Agent and Mediation device. The SNMP proxy agent uses SNMPv3 which provides a comprehensive security framework which guarantees that the solution is not vulnerable to most of the security violations. The framework uses SNMPv3 context names to differentiate the Network Elements from which information is required. The proposed framework makes use of a standard open source Net-SNMP package with an unique idea of mediation device which bridges Net-SNMP agent and different types of Network Elements. The mediation device is a separate software module which actually communicates to Network Elements by converting SNMP requests into proprietary protocol messages and vice versa. The proposed generic framework is implemented using Java, hence provides platform independence. The proposed framework has been validated in the Very Small Aperture Terminal (VSAT) communication network where large number of Network Elements are heterogeneous in nature.**

***Index Terms*— TCP/IP, SNMP, Proxy Agent, NE, NMS, MIB, Net-SNMP, AgentX, SMUX, VSAT**

Manuscript received September 14, 2008.

Santosh Chavan is with NMSWorks Software Pvt Ltd, Chennai-600036 India. (Phone: +91-44-22570433; e-mail: santosh@nmsworks.co.in).

R. Madanagopal is with NMSWorks Software Pvt Ltd, Chennai-600036 India. (Phone: +91-44-22570433; e-mail: madan@nmsworks.co.in).

I. INTRODUCTION

Remote management of Network Elements (NEs) in any type of communication network is quite desirable. It allows a centralized management of the Network Elements without having to deploy people and devices to the location of the elements. Furthermore, it provides a common interface to manage the element even if each type of element has a vendor specific interface. Thus remote management can save the network operator's time and money.

There are various ways by which remote management can be achieved. One way is by using Terminal Emulation (Telnet) [1]. Telnet provides remote access and allows management commands to be entered utilizing a Command Line Interface (CLI). Several problems persist even if Telnet is enabled on the NE side. Lack of robust security mechanisms on the NE side while using Telnet leads to security breaches.

Another powerful and most widely used way to achieve remote management is by employing the Simple Network Management Protocol (SNMP) [2] [3]. SNMP provides remote management by transferring messages, called protocol data units (PDUs), between the NEs and a Network Management System (NMS). The translation of the response to the request is being done by the SNMP agent, an active software module which resides on each Network Element. The non-SNMP and legacy devices widely exist in any type of network and management of such Network element is very critical.

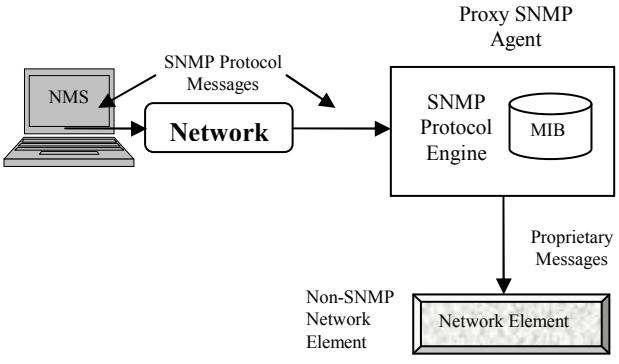

Fig.1 A typical SNMP Proxy Agent Architecture

But, how can we manage the elements that do not support SNMP? The answer to this question is a concept of Proxy Agent. A proxy agent performs the translation of SNMP requests into non-SNMP requests and vice versa [4], [5]. Fig. 1 shows standard SNMP proxy agent architecture. The messages between SNMP protocol engine and NEs are non-SNMP messages and are proprietary protocol messages. NEs which do not support Internet Protocol (IP) addressing scheme but can support a different types of communication protocols such as serial communication (RS232, RS422, and RS485) which may not be same as SNMP, Transaction Language 1 (TL1) or CLI are termed as Heterogeneous Network Elements.

To cater the management of the heterogeneous NEs, existing open source libraries can be used to develop and customize an SNMP agent/proxy agent. The one such most popular and standard open source package is Net-SNMP - an extensible SNMP agent [6]. Net-SNMP is a suite of software for using and deploying the SNMP protocol (v1, v2c and v3). The Net-SNMP agent can work in Master-Sub agent mode with:

1) AgentX [7],
2) SMUX [8],
3) Proxied SNMP Agent [3].

All three are protocols that can be used to make agents appear as one application. In each case, one agent takes the role of "Master", and delegate requests to one of the other subagent. The difference between them mainly relate to how data is represented, and the mechanism for communication between Master and subagents.

In AgentX framework, Master agent has no access to management information, whereas subagents have and are also shielded from SNMP messages from Master agent. Master agent accepts AgentX session and accepts the requests for registered MIB on which Subagent performs operations and sends response to Master agent. The basic inadequacies like lack of access control mechanism [7], congestion in channel resulted due to frequent requests [9] and security problems limits use of AgentX.

In SMUX framework, uses two approaches, namely request-response model and cache-ahead model. In the request-response model, SNMP agent simply propagates SNMP requests to user processes which exported different MIB modules and peers sends back response after performing action on requests. In cache-ahead model a peer driven model, periodically update subtree of MIB module which has been registered with agent. The registration of subtree deals with addition of MIB objects contained by subtree to the Agent's MIB. The fundamental drawbacks like priority based registration of same subtree and rejection of any operation resulting in service denial from other peers [8] makes it not a good choice in management of network elements.

In Proxied SNMP framework, typically subagent listens on non standard port, simply receives requests from Master agent and sends response back to Master agent and allows setting up access control mechanism for subagents.

The wide deployment of an extensible SNMP agent, coupled with the lack of compatibility with Internet standards in this area, makes it difficult to use them in SNMP-manageable applications. A vendor may have to support several different subagent environments in order to support different target platforms. It has also become quite cumbersome to configure subagents and possibly multiple agents on a particular Network.

To address these problems and to support heterogeneous NEs we propose a Generic SNMP proxy agent utilizing Proxied SNMP framework that supports multiple NEs. The proposed framework supports multi vendor NEs and proprietary protocol of such NEs. The proposed framework has been implemented using Java, and Net-SNMP v5.3.2 package, thereby bringing in platform independence.

II. PROPOSED GENERIC SNMP PROXY AGENT FRAMEWORK

The proposed Generic SNMP Proxy agent is a three layer framework and uses a standard open source Net-SNMP agent (v 5.3.2) as Master. First layer consist of Net-SNMP Master agent, second layer is a Mediation Device and third layer consists of all heterogeneous NEs. The work focuses more on management of multiple heterogeneous NEs and by virtue of which overcoming the drawbacks associated with AgentX and SMUX protocols listed in previous section. The framework is best suited when a network consists of every element as non-IP elements having varieties of proprietary protocols. The role of the Mediation Device is to bridge the differences between the SNMP and non-SNMP proprietary protocols. The framework uses SNMPv3 *contextNames* mechanism whereby an agent can support parallel versions of same MIB or different set of MIBs, which also helps an agent to distinguish every Network

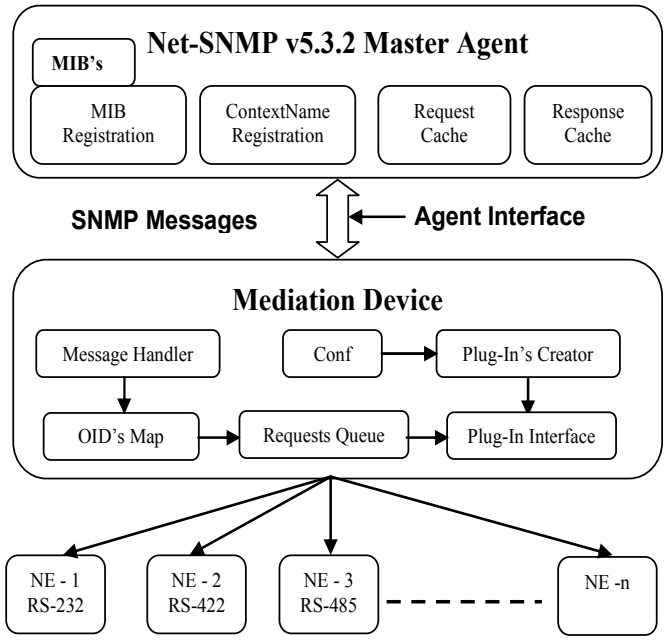

Fig. 2 A Generic SNMP Proxy Agent framework

Element from each other and their respective requests.

To implement the Generic SNMP Proxy Agent Framework, following four major components are developed. Fig. 2 shows a Generic SNMP Proxy Agent framework.

1. Net-SNMP agent as Master agent
2. Agent Interface
3. Mediation Device
4. Device Communicators.

From the NMS point of view, this framework can behave as extensible agent or appear working exactly as non-extensible (monolithic) agent. Therefore Master agent and Mediation Device can work on same machine or different machine, serving the common goal of managing heterogeneous NEs.

1. Net-SNMP agent as Master agent

The First layer of framework is Net-SNMP agent; which is extended to function as Master agent. The Master agent supports SNMPv3 which adds the mechanism to monitor and control specifically security capability of the network in conjunction with SNMPv2c. To set up SNMPv3 agent, the *snmpd.conf*, a configuration file of Net-SNMP agent needs to be configured with a *snmp engineID*. The default value of the SNMP *engineID* is configured with the IP address found for the hostname of the machine. Along with SNMP *engineID*, a *username*, an *authentication type*, *passphrase* or *privpassphrase* needs to be configured. The authentication types can be Message Digest Algorithm (MD5) or Secure Hash Algorithm (SHA). In order to use SHA authentication, Net-SNMP package needs to be built with Open SSL installed. If the privacy passphrase is not specified, it is assumed to be the same as the authentication passphrase and currently only privacy protocol supported is Data Encryption Standard (DES). The minimum passphrase length should be 8 characters. All Security related processing occurs at the message level, i.e. SNMPv3 specifies a User Security Model (USM) that makes use of fields in the message header. For the purpose of access control and distinction between NEs, each SNMP entity is considered to manage a number of contexts of managed information, each of which has a *contextName*, which is unique per NE. Access control is governed by the specific *contextName* for which access is attempted and the identity of the user requesting access.

In order to manage the NEs, the Master agent keeps the information of *contextNames* and Management Information Base (MIBs) of all the NEs. Master agent sends the request to Mediation Device to establish a connection through Agent Interface. Upon connection establishment with the mediation device, a Master agent registers list of *contextNames* and MIBs of different NEs. When Master agent receives the request from NMS, it caches the requests and assigns the *requestID* to each request and simply forwards them to Mediation device. On reception of response from the mediation device, Master agent correlates every response with cache of requests based on requestID, if match found, response is packed as SNMP PDU and sent to NMS. Master agent also maintains the response cache exclusively for *snmpwalk* and multiple queries for same OID from different users at the same time. By virtue of which it facilitates a request optimization and uses available bandwidth judiciously. Within the cleanup time of response cache, if any request arrives on same MIB tree, the response is provided from response cache avoiding traffic on the network. The cleanup time of response cache can be set to the desired value. The traps are received from Mediation Device on a dedicated trap port and sent to NMS.

2. Agent Interface

The Agent Interface is a Transmission Control Protocol (TCP) socket on non standard TCP port. The interface has two different ports assigned, one for SNMP request-response and other for Traps or Notification from NEs. On Agent Interface, for SNMP messages the Mediation device act as server and client for traps. The Agent Interface enables the communication between the Master agent and the Mediation device. The Agent Interface is used to perform following common functionalities: i) establish/close connection, ii) request SNMP requests from Master agent and forwarding to Mediation device, iii) sending response to Master agent, and iv) sending Traps to Master agent. The requests forwarded by Master Agent to Mediation Device are TCP packets.

The Agent Interface is closed if either of Mediation Device or a Master Agent fails. The Agent Interface reopens on reception of connection request by Master Agent.

3. Mediation Device

The Mediation Device is a software module which bridges the Master agent and different NEs. Each of the Network Element communicates to the Mediation Device through the bridge on its proprietary protocol. Mediation device has four major modules as follows,

I. Message Handler – Handles request and response

II. Plug-In Creator – Creates Individual threads for communicating to every NE

III. OIDs Mapping – Requested OID mapped onto respective protocol of Network Element.

IV. Plug-In Interface - Facilitates dynamic start, stop or restart of an individual plug-in

Mediation device initializes itself after reading a *conf* file. Configuration file is an Extended Markup Language (XML) file which contains the details of each NE such as *device type*, device name, *contextName*, and request queue size. Any new element addition can be done by adding the configuration details mentioned above in the *conf* file. A manual element addition also helps to control addition of malicious subagents

and avoids a chance of security breach.

On parsing a conf file, a *Plug-In creator* will create the Plug-Ins, and numbers of Plug-Ins are equal to the number of entries of NEs in the *conf* file. Each Plug-In initializes itself and starts listening. Meanwhile, on successful establishment of connection with the Master agent, each Plug-In report the *contextNames* and device type associated with them to the Master agent which registers them. Connection with a Master agent allows the transfer of requests from the Master agent. The request–response mechanism is handled by means of *contextNames* only. Consider Master Agent maintains a different set of MIBs pertaining to each type of non-IP NEs. The mediation device which bridges Master agent and NEs collects and maintains management information on behalf of each Network Element, according to unique contextName set up. When NMS wishes to access management information pertaining to individual Network Element, it specifies an appropriate *contextName and Device Type*. When the same Network Element has multiple instances, Mediation Device maintains different and unique *contextNames* pertaining to same MIB of Network Element. And data collection is done based on the *contextNames* again

A *Message Handler* scheme can send request to Master agent for request. The *Message Handler* sends the requests to *OID Mapping* for mapping an individual request into a Network Element specific protocol request. *Message Handler* forwards response from each NE to Master agent and traps are forwarded to through a trap port assigned on Agent Interface. The mapped requests are queued up based on the *contextName* and device type for sending on the respective Plug-Ins. The Queue will be unbounded concurrent thread-safe queue based on linked nodes. This queue orders elements FIFO (First-In First-Out). The head of the queue is the element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and queue retrieval operations obtain elements at the head of the queue. Such type of queues is appropriate choice when many threads share the common collection. This queue does not permit null elements. Determination of current number of elements in queue requires traversal of the elements through queue because it is asynchronous in nature.

Since the NEs are non-SNMP elements hence they can not report the exception i.e. faults associated with NE on their own to the NMS. Therefore faults in the NEs are categorized based on severity; the typical severities are critical, major, minor, and warnings. All the critical faults are processed as Traps or notifications and passed to the Master agent who in turn sends them to the NMS.

Each Plug-In necessarily implements an interface which facilitates a dynamic start, stop or restart of an individual plug-in. In case of failure of any plug-In, it allows to restart a failed plug-In without stopping a Mediation device thereby providing

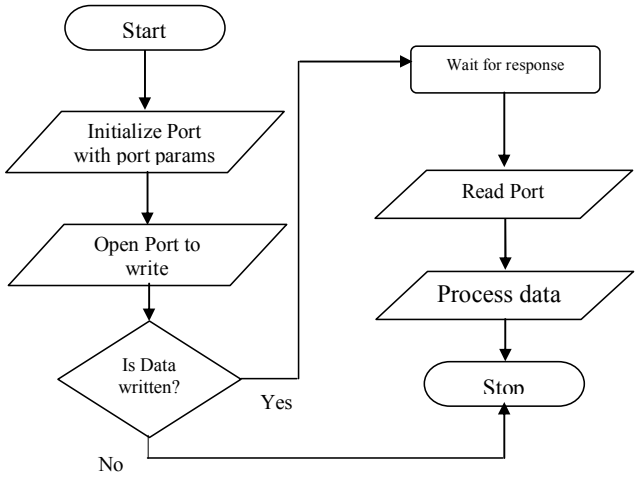

Fig. 3 A Typical Algorithm employed in Serial

a seamless data transfer from other NEs and reduce a downtime of the network.

The NE specific requests from queue are sent to the respective Device communicators. The Mediation device accepts a NE protocol specific response does a processing on the response if required and packs the processed response as TCP packet and it is sent to the Master agent.

4. Device Communicators

Device communicator is the last layer in the framework and is a software module which actually communicates with the NEs using their proprietary protocols. This module is an external layer to Master agent – Mediation Device. Every Network Element is connected to the Generic SNMP Proxy Agent for its management. The communicator reads the mapped protocol requests and gets the response from the NEs. The received response is processed locally if required.

Considering a NE which supports RS-232 based serial communication protocol for monitoring and control. Typically a Serial communication algorithm would look like as depicted in Fig. 3. For serial communication, a port is initialized by setting up specified port parameters such as baud rate, start bit, stop bit, and parity bit. Upon initialization, the specified port is opened for writing a command or a network specific protocol frame. The Network Element responds if the port settings and received command or Network Element specific protocol is correct. After the end of the data transfer or on communication failure the port can be closed.

III. EXPRIEMENTAL VALIDATION OF THE PROPOSED GENERIC SNMP PROXY AGENT

In order to validate the proposed Generic SNMP Proxy Agent Framework, we had a Very Small Aperture Terminal (VAST) network under management as a case study. Fig. 4 depicts the VSAT network under management. The VSAT network has non-IP NEs such as L-Band Up/Down Frequency

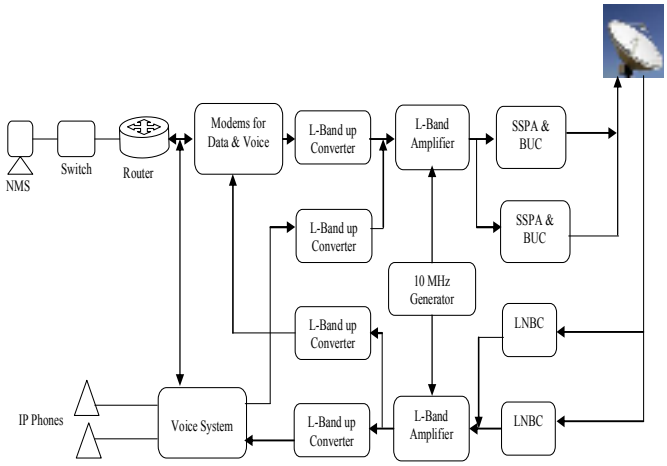

Fig. 4 A VSAT Network under Management

converters, L-Band Amplifier, , Block up Converters (BUC), Solid State Power Amplifiers, Frequency Generators, Antenna Controller Unit (ACU), Beacon Tracking Receiver (BTR), Low Noise Amplifiers (LNBC) [10]and Modem Redundancy Controllers[11] etc. The listed NEs have variety of physical interfaces for communicating on the network such as RS-232, RS485, RS-422 or Telnet. Totally 11 non-IP NEs are connected to the Serial to Ethernet converter (Moxa-5610) [12], which provides a standard Ethernet interface for communicating with the NMS. Each port of Serial to Ethernet converter is assigned to each Network Element.

The Generic SNMP Proxy Agent is exactly analogous in nature to the Serial to Ethernet Interface. When NMS requests for management information from particular Network Element, a request is passed through a Mediation Device to the Network Element, which in turn passes through on assigned Serial to Ethernet port. The MIBs for each of the Network Element are written in Abstract Syntax Notation dot One (ASN.1) format. The master agent is configured with all MIBs pertaining to each Network Element. The entire system work seamlessly and the data from each Network Element on the network is acquired. The performance related data are represented using HTML as well as Java Client. The Fault manager is used to handle faults and traps/notifications.

IV. PERFORMANCE OPTIMIZATION

Most of the heterogeneous NEs, communicate with the network using various serial communication interfaces. Such NEs work at specified baud rates, ranging from 9600- 38400 bps. Lower the baud rate higher the response time and vice versa; therefore low baud rates affect the performance of the data transfer between NMS and NEs. And also the lower baud rates introduce a significant amount of delay in acquiring response to an individual request. To address this problem and improve the performance of data transfer between NMS and NEs, the request-response mechanism has been optimized in Generic SNMP Proxy Agent. Following are the two different scenarios considered where request-response mechanism is improved: i) when multiple users (clients) request for the same management information i.e. one OID or multiple OID's at the same time, and ii) when a subtree of MIB is subjected to *snmpwalk* command.

In first case, consider n numbers of clients are seeking management information simultaneously from same Network Element. Assuming t is time taken by the Network Element to respond to first client. A second client has to wait for 2t amount of time, and similarly n^{th} client have to wait for (n x t) amount of time. This process adds a large delay in providing response to all clients. In the scenario described, the request queue in Mediation Device is classified based on same types of requests. And for such classified requests, only one request is executed on behalf of class of same requests. The response received to one request is distributed among the 'n' clients.

In Second case, when a subtree of MIB of a particular Network Element is subjected to the *snmpwalk* command by NMS. The response of the *snmpwalk* has been cached in the Master agent. A response cache is maintained for a desired amount of time which is called as clean up time of cache. After the cleanup time, response cache is flushed out and response of next *snmpwalk* is again cached. Within the cleanup time of response cache, if any client is seeks management information on the same subtree, response to that particular request is taken from the response cache and sent to the NMS, instead of sending the request to the Network Element.

Both these mechanisms not only avoid congestion on the network because of avalanche of requests but also improve the performance of the Generic SNMP Proxy Agent to a great extent.

V. CONCLUSION

The developed Generic SNMP Proxy Agent framework operates with any type of Network Element from different vendors, equipped with its proprietary protocols. It also provides the flexibility to add them in the network. The developed framework assures a seamless communication and management of multiple heterogeneous NEs. The use of SNMPv3 and its *contextname* mechanism provides an access control mechanism and a comprehensive security on both data transfer and registration of sub-agents. The query optimization and Response -cache phenomenon helps to limit the congestion in the channel on multiple queries and improvises a response time significantly. The framework maintains the list of MIBs and allows those listed MIBs and its OIDs to register as many as times required with Proxy Agent thereby giving no space to registration of any malicious subtree. The Generic SNMP Proxy Agent also facilitates the dynamic addition or deletion of Plug-Ins when failed or required. Thus a proposed Generic SNMP Proxy Agent framework overcomes the drawbacks associated with AgentX and SMUX and makes it more apt for applications where hundreds of heterogeneous elements are to be managed. Thus implementation of the proposed framework ensures total network management solution for any set of heterogeneous NEs.

REFERENCES

[1] Web page, [Online] , Available: Telnet http://tools.ietf.org/html/rfc15

[2] Mani Subramanian, Network Management: Principle and Practice, 1st ed. Reading, MA: Addision-Wesley, Jan 2000.

[3] W. Stallings, SNMPv1, SNMPv2 and SNMPv3, 3rd ed. Reading MS: Addision-Wesley, Dec. 1998.

[4] Web page, [Online] , Available: RFC 1208, http://www.ietf.org/rfc/rfc1208.txt

[5] Web page, [Online], Available: RFC 2273, http://www.ietf.org/rfc/rfc2273.txt.

[6] Web page, [Online] , Available: Net-SNMP Extensible Agent, http://www.net-snmp.org

[7] Web page, [Online], Available: AgentX, http://www.ietf.org/rfc/rfc2741.

[8] Web page, [Online] , Available: SMUX, http://www.ietf.org/rfc/rfc1227.txt

[9] Rong Jin, Weiming Wang, "Research and Implementation of SNMP in ForCES Framework", International Conference on Wireless communication, Networking and Mobile Computing, 21-25 Sept. 2007, pp3106-3109.

[10] Web page, [Online], Available: www.gdsatcom.com/vertesrsi.php.

[11] Web page, [Online], Available: http://www.radyne.net.

[12] Web page, [Online], Available: MOXA 5610, http://www.moxa.com/product/nport_5600.

Santosh S. Chavan was born in pune, Maharashtra, India on October 16, 1981. He received the B.E. degree in Instrumentation Engineering from SRTM University, Nanded Maharashtra, India in 2002.and M.S. by Research in Electrical Engineering, from Indian Institute of Technology, (I.I.T.) Madras Chennai India in 2008. He is currently a Sr. Software Engineer in NMSWorks Software Pvt. Ltd. He is currently working in the area Integrated Network Management Systems for Defense communication networks. His research interests are in the areas of Computer Networks, Wireless Networks, Network Management Systems and Digital Design.

R. Madanagopal received B.E degree in Computer Science and Engineering from Bharathiar University, Coimbatore, India, in 2002 and M.S. degree in Computer Science and Engineering from Indian Institute of Technology, (I.I.T.) Madras in 2008. He is currently a Technical Architect in NMSWorks Software Pvt. Ltd., Chennai. He is working in the project on Advanced Network Systems which deals with Integrated Network Management Systems (NMS) for different types of Networks. His research interests include service provisioning in transport networks, networking, algorithms and distributed systems.