KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by SHAIK SIRAJUDDIN under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

_____
Dr. Mohammed H Sqalli (Advisor)

_____
Dr. Mayez Al Muhammed (Member)

_____
Dr. Uthman Baroudi (Member)

_____
Dr. Abdul Aziz Al Mulhem
(Department Chairman)

_____
Dr. Mohammad Al- Ohali
(Dean of Graduate Studies)

_____
Date

ACKNOWLEDGEMENT

All praise and thanks are due to Almighty **Allah,** *subhana-wa-taala,* the most Merciful; the most Benevolent, for bestowing me with the health, knowledge, opportunity, courage and patience to complete this work. Thereafter, acknowledgement is due to KFUPM for the support given to this research through its tremendous facilities and for granting me the opportunity to pursue graduate studies with financial support.

I acknowledge, with deep gratitude and appreciation, the inspiration encouragement, valuable time and guidance given to me by my thesis Committee Chairman, Dr. Mohammed Houssaini Sqalli.

Thereafter, I am deeply indebted and grateful to Dr. Mayez Al Muhammed and Dr. Uthman Baroudi, my committee members, for their extensive guidance, continuous support, and personal involvement in all phases of this research.

Special thanks are due to my friends Mazher, Abdul Qaiyyum, Baqtiyar, Yousuf, Kashif, Ayub Azher, Hafeez, Basha, Khaja, Obaid, Rizwan Baba, Abbas, Ismail, Saqib, Mujeeb, Humayun Baig and all others who provided wonderful company and good memories that will last a life time. My special thanks to my uncle Mohammed Anwar and his sweet little daughter Mariyam.

Family support plays a vital role in the success of an individual. I am thankful to my entire family for their love, support and prayers throughout my life especially my dearest mother, grandmother and my wife.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

THESIS ABSTRACT (ENGLISH)

Name: Shaik Sirajuddin

Title: Extensions To XML-based Network Management

Major Field: Computer Engineering

Date of Degree: January 2005

XML-based integrated network management architecture consists of an XML-based manager, an XML/SNMP gateway and SNMP agents. In this thesis, we present frameworks for extensions to an existing XML-based network management, which can reduce the processing time between the XML-based manager and the SNMP agents. The extensions consist of new types of messages, including Multi-Get-Request and Multi-Set-Request. These new types, for instance, allow a manager to send one or more requests to one or more agents. We proposed three types of frameworks for the XML-based network management namely Single DOM (Document Object Model) Tree-based approach, CSV (Comma Separated Values) -based approach, and JPVM (Java Parallel Virtual Machine) based approach. We present three JPVM work assignment methods for parallel network management namely equal work for every JPVM gateway, Static weighted load balancing based on processing capability of the JPVM gateway and dynamic load balancing for heterogeneous network of stations. We have evaluated and compared our framework with other frameworks. Our approach reduced the time by ~48%, ~71%, ~85% respectively for CSV, JPVM with parallel tasks and JPVM with Distributed processors.

ملخص الرسالة

الاســـــــــم :شيخ سراج الدين

عنوان الرسالة :إضافات و إلحاقات جديدة لإدارة الشبكات المبنية على تكنولوجيا XML

التخصـــــص : هندسة الحاسب الآلي

تاريخ التخـرج : يناير2005

تتألف هيكلة إدارة الشبكات المتكاملة القائمة على تكنولوجيا XML من حاسوبٍ مديرللشبكات قائم علىXML ، منفذ XML/SNMP، وعناصر مستخدمة لبروتوكول SNMP. في هذا البحث، نقدّم هيكلاً لإضافات و إلحاقات جديدة في إدارة الشبكات الموجودة حاليا والقائمة على تكنولوجياXML ، والذي يمكن من تخفيض وقت المعالجة بين الحاسوب المدير للشبكات القائم على XML والعناصر المستخدمة لبروتوكول SNMP. تتألف هذه الإضافات والإلحاقات من أنواع جديدة من الرسائل بما في ذلك طلب-تحصيل -متعدّد و طلب-تغيير-متعدّد .من خلال هذه الإضافات والإلحاقات الجديدة، يمكن مثلا لمدير الشبكات إرسال طلب واحد أو أكثر إلى عنصر SNMP واحد أو أكثر .نقترح في هذا البحث ثلاثة أنواع من الهياكل لإدارة الشبكات القائمة على تكنولوجيا XML وتشمل :الطريقة القائمة على شجرة DOM المنفردة، الطريقة القائمة علىCSV ، والطريقة القائمة على آلة فعليّة متوازية قائمة على بروتوكول JAVA (JPVM). كما نقدّم في هذا البحث ثلاثة طرق لتعيين واجب العمل لإدارة الشبكات المتوازية وتشمل :العمل المتساوي لكلّ منفذJPVM ، الموازنة الثابتة للعبء المرجّح القائمة على قدرة المعالجة لمنفذJPVM، والموازنة الديناميكيّة لشبكة غير متجانسة من المحطّات .وفي نهاية هذا البحث، قمنا بتقييم ومقارنة هيكلنا بهياكل أخرى.

CHAPTER 1

INTRODUCTION

Today's network has incompatible infrastructure including different information models, information access methods, and management protocols. The administrator has no choice but to use separate and incompatible management tools to manage the current heterogeneous network. Currently available management tools and framework are based on a centralized approach and confronted with scalability and efficiency problems when the network expands.

When Java applets appeared in Netscape's famous web browser [1] [2], in 1995, it introduced the concept of *embedded management application*, and has the advantages of using HTTP rather than SNMP to vehicle data between managers and agents. In 1996, The Simple Times [3] reported different ways of integrating the HTTP, HTML, and applets with standard IP network management platforms. The network management companies and customers started using the web-based management interface with the use of web browsers to display management data using Graphical User Interfaces (GUIs). Managing the network components using web-based [3] [4] technology came into existence when the vendors began embedding HTTP servers in their network equipment. Many network equipment vendors, including Cisco, Nortel Networks and 3Com, now routinely embed HTTP servers in their new equipment.

XML-based [5] network management applies XML technologies to network management. In XML-based network management, the management information is defined using XML and the management data is exchanged in the form of an XML document and processed using the standard methods available for XML document processing.

In this section, we give a general background of network management, our problem statement, and the thesis layout.

1.1. NETWORK MANAGEMENT

Network management models consist of four components, Network Management Stations (NMSs) or Manager, agents running on managed nodes (Managed nodes can be router, switch, pc, Unix server etc.), management protocols, and management information. A manager is a server running some kind of software system that can handle management tasks for a network. Managers are often referred to as *Network Management Stations* (NMSs). An NMS is responsible for polling and receiving traps from agents in the network. Agent is a piece of software that runs on the network devices we are managing. It can be a separate program (a daemon, in Unix language), or it can be incorporated into the operating system (for example, Cisco's IOS on a router, or the low-level operating system that controls a UPS). Today, most IP devices come with some kind of SNMP agent built in. The agent provides management information to the NMS by keeping track of various operational aspects of the device. For example, the agent on a router is able to keep track of the state of each of its interfaces: which ones are up, which ones are down,

etc. The NMS can query the status of each interface on a router, and take appropriate action if any of them are down. When the agent notices that something bad has happened, it can send a trap to the NMS. This trap originates from the agent and is sent to the NMS, where it is handled appropriately. Some devices will send a corresponding "all clear" trap when there is a transition from a bad state to a good state. An NMS uses the management protocol to communicate with agents running on the managed nodes. The *Structure of Management Information* (SMI) provides a way to define managed objects and their behavior. An agent has in its possession a list of the objects that it tracks. One such object is the operational status of a router interface (for example, *up*, *down*, or *testing*). This list collectively defines the information the NMS can use to determine the overall health of the device on which the agent resides.

The *Management Information Base* (MIB) can be thought of as a database of managed objects that the agent tracks. Any sort of status or statistical information that can be accessed by the NMS is defined in a MIB. The SMI provides a way to define managed objects, while the MIB is the definition (using the SMI syntax) of the objects themselves. Like a dictionary, which shows how to spell a word and then gives its meaning or definition, a MIB defines a textual name for a managed object and explains its meaning.

An NMS collects real time data from network elements such as routers, switches, and workstations. It interprets and analyzes the data collected, and presents this information to authorized network operators. In addition, it proactively reacts, in real time, to management problems.

1.2. NETWORK MANAGEMENT MODELS

The most important two network management models are the *pull model* and the *push model* used for exchanging data between two distant entities [1]. The pull model is based on the request/response paradigm (called *data polling,* or simply *polling,* in the SNMP management framework). The client sends a request to the server (i.e. agent), then the server answers, either synchronously or asynchronously. This is functionally equivalent to the client "pulling" the data off the server. In this approach the data transfer is always initiated by the client (i.e. manager). The push model is based on the publish/subscribe/distribute paradigm. In this model agents first advertise what MIBs they support, and what SNMP notifications they can generate. The administrator then subscribes the manager (i.e. NMS) to the data he/she is interested in, specifies how often the manager should receive this data and disconnects. Later on, each agent individually takes the initiative to "push" data to the manager, either on a regular basis or via a scheduler. The advantages of using the push model are to conserve network bandwidth and move part of the CPU burden from managers to agents.

With the push model, the manager contacts each agent once, subscribes to an OID once (push data definition), and specifies at what frequency (push frequency) the agent should send the value of this OID (push data schedule). The push model introduces a new issue: synchronization. If the manager and the agent have internal clocks that do not synchronize regularly then they will probably drift apart.

Our approach with SNMP management framework is based on the request/response paradigm, which is a *pull model*.

1.3. PROBLEM STATEMENT

The SNMP-based network management has limitations [6] [7] [8] [9] such as scalability, efficiency, and large amount of data transfer. XML-based network management was proposed to overcome some of these limitations. But the current XML-based network management suffers from the following problems.

- Managing multiple network devices, i.e., sending XML-based request to a set of SNMP agents is still not addressed.
- No generalized framework for the XML-based network management.
- Processing efficiency of the XML-based request, i.e., the time taken to process the XML-based request is high.
- The XML/SNMP gateway results in an unexpected delay between managers and agents, which might become a bottleneck when the network expands in the future.

In our extensions to XML-based network management, we propose to enhance on the exiting XML-based network management. In this work, we provide a way to manage multiple network devices. We present our framework to overcome the processing overhead of the XML-based request. Then, we evaluate the performance of the proposed framework and compare it with exiting frameworks. In our proposed extensions to the existing XML-based network management, the manager can send more advanced requests

to the agents via an XML/SNMP gateway. A manager can, for instance, send one request to multiple agents, multiple requests to one agent, or a combination of both.

Figure 1.1 shows the SNMP-based request, where the manager sends an SNMP-based request and receives the corresponding SNMP-based response. The general format of the SNMP request $(SNMP-request = Agent, MIB_1, MIB_2, ...MIB_n)$ consists of an agent name followed by a list of MIB objects requested from that agent. The traffic between the manager and the SNMP agents increases as the number of SNMP-agents grows.

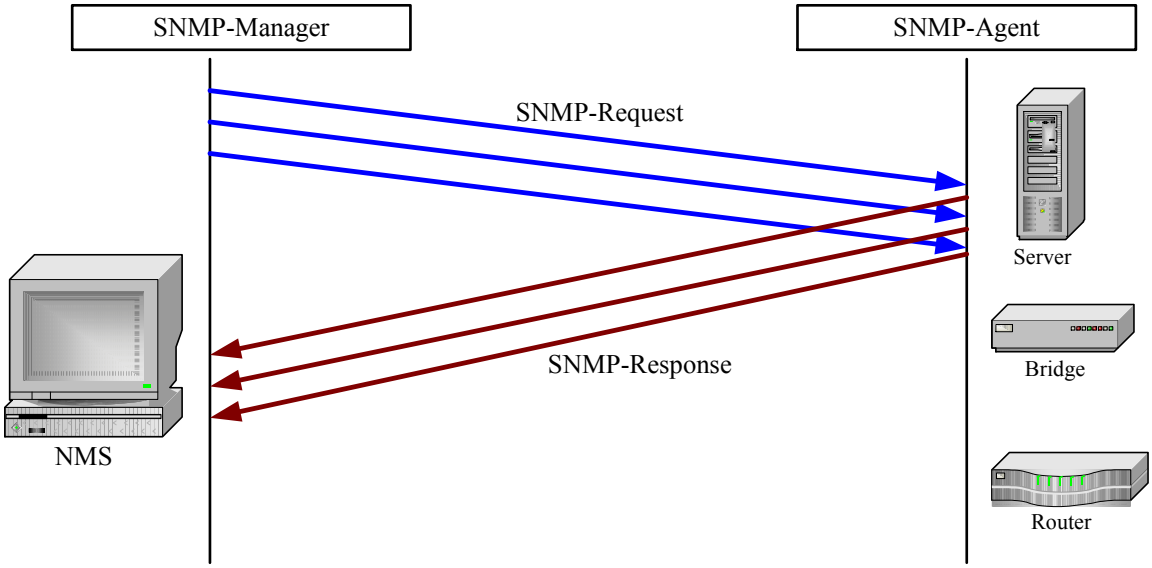

Figure 1.1: SNMP-based Request

Figure 1.2 shows the extensions to the XML-based request, where the XML-based manager communicates with SNMP agents via an XML/SNMP gateway. The format of the extensions to the XML-based request

$(XML-request = Agent_1, Agent_2, ...Agent_k, MIB_1, MIB_2, ...MIB_n)$ consists of a list of agents followed by another list of MIB objects requested from the agents. Hence, the manager can send a single request to multiple agents. This reduces the traffic between the XML-based manager and the XML/SNMP gateway.

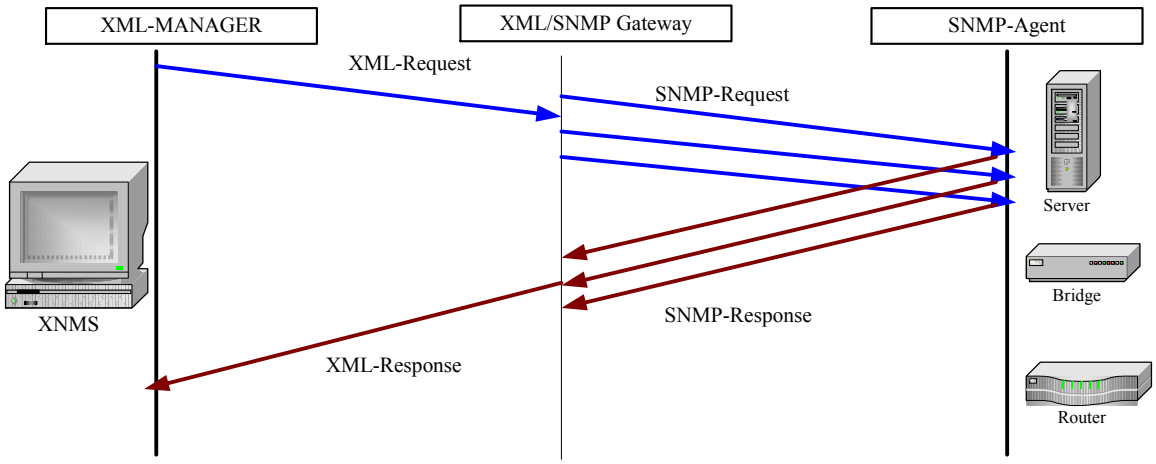

Figure 1.2: XML-based Requests

1.4. THESIS LAYOUT

The thesis is organized as follows; Chapter 2 will address the limitations of the traditional SNMP-based network management and describes the XML-based technologies with respect to network management. Chapter 3 will present the current work on XML-based network management. Chapter 4 will describe our proposed frameworks for the extended XML-based network management. Chapter 5 will present the evaluation results and comparison with previous work. Finally we conclude our work in Chapter 6.

CHAPTER 2

BACKGROUND

2.1. SNMP-BASED NETWORK MANAGEMENT

The Simple Network Management Protocol (SNMP) is the most widely used protocol to manage network devices on the Internet. The Internet Engineering Task Force (IETF) first standardized SNMP in 1990 [RFC 1157] [7] [8] [9]. A number of Requests for Comments (RFCs) have been written to specify the different elements and versions of SNMP. SNMP uses a general manager and agent interaction model (Request/Response). It uses the Structure of Management Information (SMI) [RFC 1155, RFC 2578] to define managed objects. The SNMP Management Information Base (MIB) [RFC 1213] uses a hierarchal tree structure for organizing the MIB Object Identifiers (OIDs). The first version of SNMP is referred to as SNMPv1. SNMPv1 supports GET-REQUEST, SET-REQUEST, GET-NEXT-REQUEST and TRAP operations, and provides limited management capabilities. SNMPv1 has few limitations including the lack of security, lack of bulk data transfer capability, and lack of manager-to-manager communication.

These issues were addressed in SNMPv2 [RFC 3416], which was initially proposed in 1995. SNMPv2 supports GET_BULK_REQUEST, and INFORM_REQUEST. The major changes in SNMPv2 are the addition of manager-to-manager message, enhancements to SMI (SMIv2) [RFC 2578], textual conventions [RFC 2579], conformance statements [RFC 2580], row creation and deletion in table [RFC 2579], MIB enhancements [RFC

3418], and transport mappings [RFC 3417]. One of the main limitations of SNMPv2 is security, which included a community-based mechanism that uses a plain text string for authentication and access control.

SNMPv3, introduced in 1999 [9], undertook the issue of security including authentication, privacy and access control, as well as the definition of new architecture and framework for SNMP [RFC 3410-3415]. [RFC 3584] described the coexistence between SNMPv1, SNMPv2, and SNMPv3.

The SNMP framework is designed to minimize the number and complexity of management functions by the agents. This makes it extensible to accommodate additional and unanticipated aspects of network operations and management, and independent of the implementation of a particular host or gateway. Thus, SNMP provides simplicity, interoperability, and low footprint on agents. SNMP has wide support of IP equipment vendors.

The SNMP-based network management is simple in nature but has few limitations. The limitations of SNMP-based network management can be broadly categorized into three.

- Scalability
- Efficiency
- Large amount of data transfer

2.1.1. Scalability

The most important drawback of the SNMP-based network management is scalability to support a large network [6] [7]. The main factor is network overhead. In an SNMP-based the NMS network overhead is the proportion of a link capacity to transfer management data. As the number of agents to be managed increases, the management data transmitted over a single communication line from all the agents to the SNMP-based manager also increases. The capacity of the manger local segment is limited due to the centralization of management [5]. Data received from all the agents is accumulated at one single point. Hence, the network management overhead must represent a small percentage of the overall capacity of the link. The network capacity must be utilized for user data transfer, and not for management data.

2.1.2. Processing Time

In SNMP-based network management, processing time is nothing but latency. It is the time taken between sending a request for the MIB variables and the time of receiving the response from the agent. The latency must be low. If it is very high then operational problems are detected very slowly and corrected lately. Latency can be divided into two [5], End-host latency and Network latency. End-host latency is due to the marshaling and unmarshaling of the data, compression and decomposition of the data, and security key computation. Network latency is the time spent in the network links and network equipments. It depends on the capacity and the error rates of the links, and on the speed of

the routers traversed between the agent and the manager. The amount of data moved on the links has direct impact on the network latency.

2.1.3. **Large Amount of Data Transfer**

The traditional SNMP-based network management can support only up to maximum message size of 1472 bytes [9] [10], and which can be transmitted over UDP protocol. In the case of XML based network management the request is text based and has large amount of bandwidth for transmission compared to SNMP-UDP packet.

2.2. XML TECHNOLOGIES FOR NETWORK MANAGEMENT

XML (Extensible Markup Language) is a Meta markup language, which was standardized by the World Wide Web Consortium (W3C) for document exchange in 1998 [11] [12] [13]. XML has many advantages for instance, we can define our own Structure of Management Information in a flexible form using either Document Type Definition (DTD) or XML Schema. XML documents can be transmitted on the Internet using HTTP (Hyper Text Transport Protocol). XML offers many free APIs for accessing and manipulating the XML data. XML separates the contents of a document and the expression methods, i.e. the management data is stored in XML documents and the presentation or format of the management data is stored in XSL (Extensible Style Sheet Language) documents using XSLT representation [12] [13]. XML supports the exchange of management data over all the hardware and software that supports HTTP. XML needs

low development cost, since all the APIs and development kits are freely available. XML supports transfer of large amount of data in a single document. All these advantages of XML make it a candidate to solve the problems of scalability and efficiency of existing SNMP based NMS. In this section we explain the XML technologies with respect to XML-based network management.

2.2.1. **XML Document**

An XML document consists of tags similar to those of a HTML document. The XML document contains only data between the tags. We can define our own tags to represent data. We can define our own data structures in way to suitable for our data representation. SNMP SMI (Structure of Management Information) can be represented in the form of an XML document.

XML is a text-based document, and we need a mechanism to structure, and validate the contents present in an XML document. W3C proposed two ways to structure the XML document contents.

- DTD (Document Type Definitions).
- XML-Schema.

2.2.2. **DTD**

DTD [13] is used to represent the structure of each element present in the XML document. The content description is part of the element declaration in DTD, and specifies the order and quantity of elements that can be contained within the element being declared. DTD is used to specify a property for each element in addition to the relationship between the elements. DTD does not support a complex information model, so we need to convert each object of SNMP MIB into its equal element. To overcome the limitations of DTD, W3C proposed another modeling mechanism, XML Schema. XML Schema substantially revised and extended the capabilities found in XML DTDs.

2.2.3. **XML-Schema**

The XML schema [14] [15] [16] is machine readable and human readable. An XML schema document is basically an XML document. XML Schema supports a variety of data types (44 kinds of basic types), while DTD treats all data as strings or enumerated strings. XML Schema also allows inheritance relationships between elements and supports namespace integration. XML schema provides modularity XML schema offers greater control and flexibility than the DTD. It is complete and more complex than the DTD model.

XML schemas are used to define the Structure of Management Information and the constraints that the MIB objects have to satisfy. SMI can be defined according to the user

requirement. Table 2.1 shows the code in ASN.1 notation of OBJECT TYPE macro in the SNMP MIB [5].

Table 2.1: ASN.1 OBJECT TYPE Macro in SNMP MIB

ASN.1 Object Type Macro
NodeName OBJECT-TYPE SYNTAX “SyntaxType” ACCESS “AccessType” STATUS “StatusType” DESCRIPTION “DescriptionText” REFERENCE “ReferenceType” INDEX “IndexList” DEFVAL “DefaultValue” : : = {parentNodeName nodeNumber}

This macro is used to represent table nodes or the data node of the MIB. The equivalent conversion of the OBJECT TYPE macro expressed in XML schema is shown in Table 2.2.

Table 2.2: XML Schema Representation of the OBJECT-TYPE Macro

XML Schema For Object Type Macro
<xsd:element name = “NodeName”> <xsd:complexType> <xsd:simpleContent> <xsd:restriction base = “xsd:string”> <xsd:sequence> (lower part node definition part) </xsd:sequence> <xsd:attribute name = “oid” type =“xsd:string” use = “fixed” value =“OidValue” /> <xsd:attribute name =“Access” type =“xsd:string”use = “fixed” value =“AccessType”/> <xsd:attribute name = “Status” type =“xsd:string” use = “fixed” value =“StatusType” /> <xsd:attribute name = “Description”type = “xsd:string” use = “fixed”value = “DescriptionText” /> <xsd:attribute name = “Reference”type = “xsd:string” use = “fixed”value = “ReferenceType” /> <xsd:attribute name = “Index” type =“xsd:string” use = “fixed” value =“IndexList” /> <xsd:attribute name =“Defval” type =“xsd:string” use =“fixed” value =“DefaultValue” /> </xsd:restriction> </xsd:simpleContent> </xsd:complexType> </xsd:element>

2.2.4. XPATH

The primary purpose of Xpath [13] [17] [18], XML Path Language, uses an expression to identify nodes in an XML document. An XPath pattern is a slash-separated list of child element names that describe a path through the XML document. The pattern "selects" elements that match the path is to address parts of an XML document. It also provides basic facilities for manipulation of strings, numbers, and Boolean. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath gets its name from the use of a path notation as in URLs for navigating through the hierarchical structure of an XML document.

XP ath models an XML document as a tree of nodes. There are different types of nodes, element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node [17].

One important kind of expression is a location path. A location path selects a set of nodes relative to the context node. The result of evaluating an expression that is a location path is the node-set containing the nodes selected by the location path. Location paths can recursively contain expressions that are used to filter sets of nodes. A location path can be absolute or relative.

If the path starts with a slash (/) it represents an absolute location path to an element. If the path starts with two slashes (//) then all elements in the document that fulfill the criteria will be selected (even if they are at different levels in the XML tree), and is a relative path.

An example of XPath is given in Table 2.3, consider an XPath "/multiget" which selects the type of operation. The XPath " /multiget/host/@name" will select all the host names from the given XML-based request. The XPath "/multiget/host/xpath/@MIB" will select all the MIB objects from the given XML-based request. The XPath "//value" will select all the values from the given XML-based request.

Table 2.3: An Example of XPath

Example of XPath

```
<?xml version="1.0" encoding="UTF-8" ?>
- <multiget>
  <version>0</version>
  <RCommunity>public</RCommunity>
  <Port>161</Port>
- <host name="172.16.104.230">
- <xpath MIB="sysDescr">
  <value>3Com SuperStack II</value>
  </xpath>
- <xpath MIB="sysContact">
  <value>netserv@ccse.kfupm.edu.sa</value>
  </xpath>
- <xpath MIB="sysLocation">
  <value>22-419</value>
  </xpath>
- <xpath MIB="sysName">
  <value>3Com419-90</value>
  </xpath>
</host>  </multiget>
```

2.2.5. XQUERY

XQuery [18] [19] is an XML Query language, is a language for finding and extracting (querying) data from XML documents, and is designed to support all type of XML data sources like structured and semi structured documents, relational databases, and object

repositories. XQuery provides a powerful and structured facility. XQuery uses Xpath as a subset and can easily express a complicated query.

XQuery also provides features such as filtering a document to produce a table of contents, joining across multiple data sources, grouping and aggregating the contents, and querying based on sequential relationships in the XML documents.

An example of XQuery is given in Table 2.4. This XQuery takes the XML-based response document is shown in Table 2.4. The XQuery will get all the agent names that are located in building “22-335-1”. The result we obtain from this XQuery is agent names “196.1.64.255”, and “196.1.64.253”.

Table 2.4: Example of XQuery

Example of Xquery	
for $x in doc("xml-request.xml")/multigethost/host where $x/value=22-335-1 order by $x/host/@name return $x/host/@name	
XML-based Response Document (xml-request.xml)	
<?xml version="1.0" encoding="UTF-8" ?> - <multigethost> - <host name="172.16.104.230"> - <xpath MIB="sysLocation"> <value>22-419</value> </xpath> - <xpath MIB="sysName"> <value>3Com419-90</value> </xpath> </host> - <host name="196.1.64.255"> - <xpath MIB="sysLocation"> <value>22-335-1</value> </xpath> - <xpath MIB="sysName"> <value>Cat3550-335-1145</value> </xpath> - <host name="196.1.64.253"> - <xpath MIB="sysLocation"> <value>22-335-1</value> </xpath> <xpath MIB="sysName"> <value>Cat3550-335-1145</value> </xpath>	- <host name="10.22.24.17"> - <xpath MIB="sysLocation"> <value>aaa</value> </xpath> - <xpath MIB="sysName"> <value>ME-231A-24</value> </xpath> - <host name="ics-abid"> - <xpath MIB="sysLocation"> <value>23-16B</value> </xpath> - <xpath MIB="sysName"> <value>ICS-ABID</value> </xpath> - <host name="coe-yousuf"> - <xpath MIB="sysLocation"> <value>RA OFFICE</value> </xpath> - <xpath MIB="sysName"> <value>COE-YOUSUF</value> </xpath> </host> </multigethost>

2.2.6. XML Parsers

2.2.6.1. DOM

The Document Object Model (DOM) [20] [21] is a programming interface for XML documents. It is also a platform and language independent interface, which allows applications to dynamically access and manipulate the content, structure, and style of the documents. The DOM represents a *tree view* of the XML document. The *documentElement* is the top-level of the tree. This element has one or many *childNodes*

that represent the branches of the tree. The DOM provides a representation of a complete XML document stored in memory, providing random access to the contents of the entire document.

The node object represents a node in the node tree. A node can be an element node, a text node, or any other of the node types. The nodeList object represents a node and its child nodes as a node tree.

2.2.6.2. SAX

The Simple API for XML [12] [13] (SAX) is an event-driven and serial-access mechanism for accessing XML documents. SAX reads the XML document in sequential order and generates an event for a specific element. Hence if the application calls are of sequential access to XML documents then the SAX parser can be much faster than DOM. But it does not provide the hierarchical information that a DOM parser provides. While accessing the XML document, the SAX parser generates events such as the start of an element and the end of an element. By capturing the event, applications can process operations on the XML document.

2.2.7. XUPDATE

XUpdate is an XML update language, which provides open and flexible update facilities to insert, update, and delete data in XML documents. The XUpdate language is expressed

as a well-formed XML document, and uses XPath for selecting elements and conditional processing.

An Example of XUpdate is shown in Table 2.5. The XUpdate makes use of XPath expression. The select attribute of the update element contains an XPath expression. In this example the update will select the sysName MIB for the host with name "172.16.134.30", and updates the value of the sysName MIB as "KFUPM-CCSE-NMG". Similarly we can have insert, delete functionality with the XUpdate.

Table 2.5: Example of XUpdate

Example of XUpdate
<?xml version="1.0" encoding="UTF-8" ?> - <xupdate version="1.0"> <update select="//multigethost/host[@name='172.16.134.30']/XPath[@MIB ='sysName']/value">KFUPM-CCSE-NMG</update> </xupdate>

2.2.8. XSL/XSLT

XML documents generally only convey information about the structure and semantics of data. They do not usually carry information about how the information is to be viewed, displayed or rendered [12] [13].

Given a particular XML document, there are different ways in which this information can be rendered or viewed. A standard called Extensible Style Language (XSL) has been proposed to address this issue. An XML style-sheet is a group of rules for transforming an XML document. These transformations are used for the purposes of augmenting XML

document data with information about how to display or view the information. It can also be used for other forms of transformation (for example defining a mapping to tab-delimited format), i.e. an XSL style-sheet contains rules which recursively map XML elements to some other structure (such as a presentation structure). XSL conforms to the XML syntax [13].

In the transformation process, XSLT uses XPath to define parts of the source document that match one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document. The parts of the source document that do not match a template will end up unmodified in the result document. Table 2.6 shows an example of XSLT. Table 2.6contains the XML response document, and XSL style sheet document. The style sheet is written to produce HTML representation of the XML response document. The result is shown in Table 2.7. The XSLT produces HTML table representation for the XML response.

Table 2.6: Example of XSLT

Example of XSLT	
XML Response Document	**XSL Style Document**
<?xml version="1.0" encoding="UTF-8" > - <multigethost> - <host name="172.16.104.230"> - <xpath MIB="sysContact"> <value>netserv@ccse.kfupm.edu.sa</value> </xpath> - <xpath MIB="sysLocation"> <value>22-419</value> </xpath> </host> - <host name="172.16.134.33"> - <xpath MIB="sysContact"> <value>yousuf@ccse.kfupm.edu.sa</value> </xpath> - <xpath MIB="sysLocation"> <value>RA OFFICE</value> </xpath> </host> </multigethost>	<?xml version="1.0" encoding="ISO-8859-1" ?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:template match="/"> <html><body> Response From Agents <table border="1"> <tr bgcolor="#9acd32"> <th>Host</th> <xsl:for-each select="multigethost/host[1]/xpath"> <th> <xsl:value-of select="@MIB" /> </th> </xsl:for-each> </tr> <xsl:for-each select="multiget/host"> <tr> <td gcolor="yellow"> <xsl:value-of select="@name" /> </td> <xsl:for-each select="xpath"> <td> <xsl:value-of select="value" /> </td> </xsl:for-each> </tr> </xsl:for-each> </table> </body> </html> </xsl:template> </xsl:stylesheet>

Table 2.7: Result of XSLT After Transformation

Host	SysContact	sysLocation
172.16.104.230	netserv@ccse.kfupm.edu.sa	22-419
172.16.134.33	yousuf@ccse.kfupm.edu.sa	RA OFFICE

2.2.9. Advantages of XML

XML has many advantages that can be summarized as follows:

- It supports structured document definitions (E.g. DTD or XML Schema).

- It can easily transfer structured documents on the Internet through HTTP protocol.
- It can be parsed using standard APIs such as DOM, and SAX.
- It separates the contents of the documents from the presentation of the data through XSL.
- It can be transformed into HTML, text or XML using XSLT.
- It supports information exchange between all the hardware and software platforms that supports HTTP.
- It needs low development cost since all the software packages are available for free.

2.2.10. **XML Manager and Agent Combinations**

Figure 2.1 shows the manager and agent combinations in XML-based network management Figure 2.1(a) shows the most widely used network management combination. Figure 2.1 (d) is a total XML-based management combination, which is an ideal network management paradigm since there is no XML/SNMP gateway. It gives the maximum benefit compared to the other network management combinations Figure 2.1 (b) and Figure 2.1 (c) show approaches that need translation from XML to SNMP through a gateway [5] [6]. Since most network devices have legacy SNMP agents installed in them, the combination in Figure 2.1 (d) is very difficult to implement in the current network environment. In order to do so, we need to deploy XML-based agents in the network devices. Figure 2.1 (c) shows the most appropriate combination to implement in the current network management framework. This, however, requires development of an

SNMP/XML gateway to exchange the messages between the XML-based network manager and an SNMP agent.

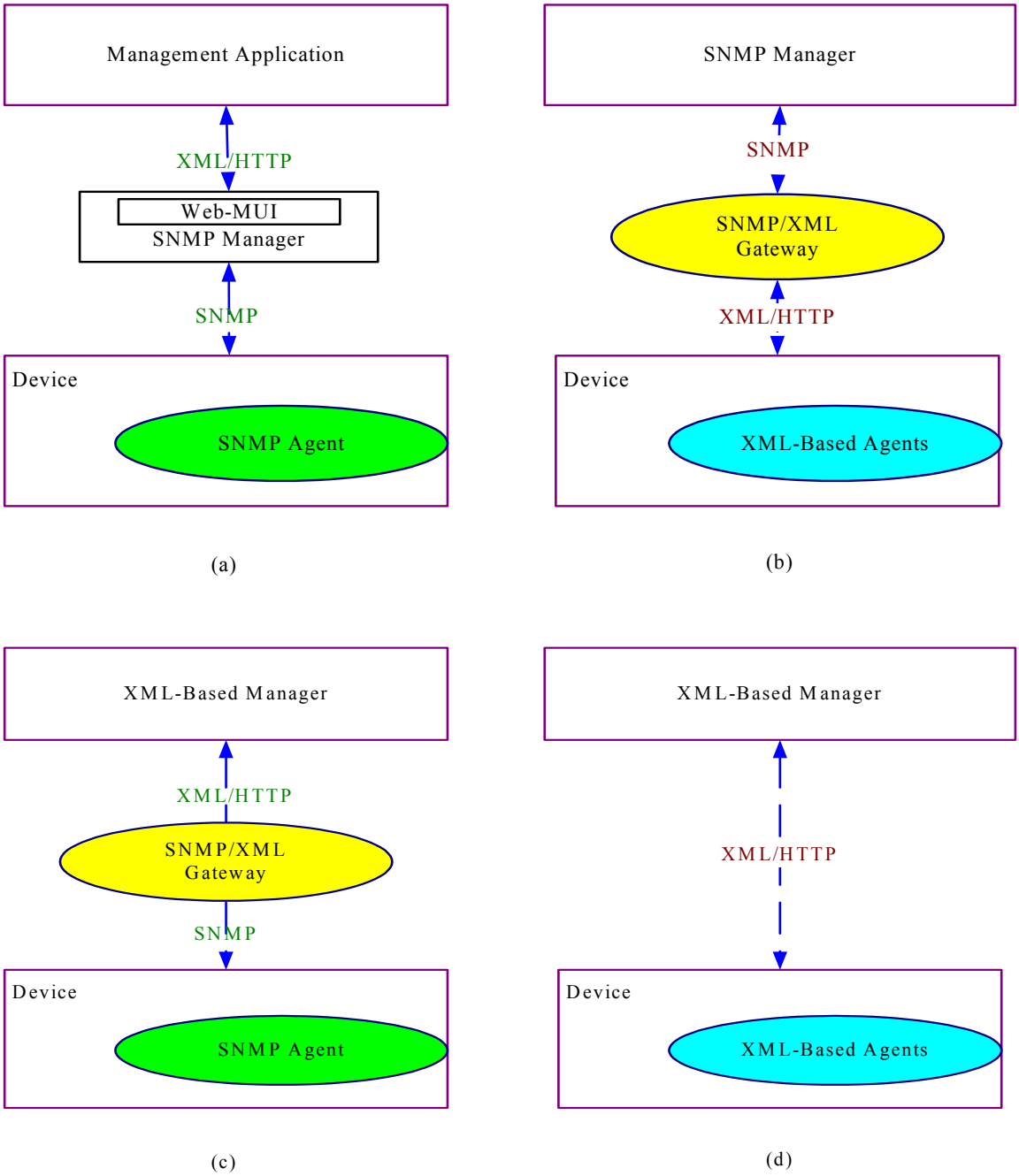

Figure 2.1: Manager and Agent Combinations in the XML-based Network Management

2.2.11. Interaction Translation Methods

2.2.11.1. Process Based Interaction Translation

DOM-based [5] interaction translation is a process-based interaction translation. In this method DOM interfaces are used for manipulate the structure for information translation. In this method interface call from the XML-based manager is translated into SNMP operation. It will be very useful when we have an internal gateway, integrated with XML-based management system. Here the manager directly accesses the management data in the DOM using the DOM API provided by the gateway.

2.2.11.2. Message Based Interaction Translation

HTTP is a message based translation method. In [5][6]HTTP-based translation method, XML/SNMP gateway translates the URI-based HTTP request from XML-based manager to SNMP requests. The URI is extended with Xpath and Xquery. The Xpath and Xquery in the URI string is used find the target objects. It is an efficient method to retrieve MIB objects in XML/HTTP communication. Examples of the URI-based request with Xpath and Xquery extensions are given in Table 2.8.

Table 2.8: Examples of using XPath, XQuery and XUpdate in HTTP Request

Example 1 of Using XPath
http://hostname:8080/gateway?XQuery=<XQuery><Query> <DeviceIP>141.223.82.72</DeviceIP><Gateway> <GatewayIP>141.223.82.56</GatewayIP> <ReadCommunity>public</ReadCommunity> <SNMPVersion>1</SNMPVersion> <MibName>RFC1213-MIB</MibName></Gateway> <XPath>device[@type="server"]</XPath></Query> <Query> … </Query><XQuery>
Example 2 of Using XQuery
<result> { Let $t := input() //ifTable/ifEntry/ ifType[contains(./text(), "6")] RETURN <totalInOutOctets count="{count($t) }"><in> { sum($t/ifInOctets/text()) } </in> <out> { sum($t/ifOutOctets/text()) } </out></totalInOutOctets> } </result>
Example 3 of Using XUpdate
<XUpdate><Query><DeviceIP>141.223.82.72</DeviceIP><Gateway> <GatewayIP>141.223.82.56</GatewayIP> <WriteCommunity>media</ WriteCommunity > <SNMPVersion>1</SNMPVersion> <MibName>RFC1213-MIB</MibName></Gateway> <Modifications><Update select="//sysContact">admin</Update><Update>…</Update> </Modifications></Query><Query> … </Query></XUpdate>

Example 2 in Table 2.8 shows the use of XQuery in the HTTP based interaction. This example finds the total number of in/out octets of the interface group. Example 3 shows the use of XUpdate to modify the MIB information present in the XML request document [13].

2.2.11.3. Protocol Based Interaction Translation

SOAP-based translation [22] is an example of protocol based interaction translation. SOAP is a protocol for exchanging XML-based messages over HTTP or SMTP. SOAP can be used as a simple messaging protocol and can be extended to an RPC protocol.

SOAP-based communication is used as a translation mechanism between the XML-based manager and the XML/SNMP gateway.

POSTECH defined three types of XML elements for the basic SOAP RPC messages between the XML-based manager and an XML/SNMP gateway. The three messages are described in Table 2.9.

Table 2.9: SOAP Messages between the XML-based Manager and Gateway

Message	Examples
Get Request	<m:getRequest xmlns:m="http://example.org/gateway"> <m:community>public </m:community> <m:version>1</m:version> <m:path>// ifSpeed[1]</m:path> </m:getRequest>
Set Request	<m:setRequest xmlns:m="http://example.org/ gateway"> <m:community>media</m:community> <m:path>//sysName</m:path> <m:value>Coe-Siraj</m:value> </m:setRequest>
Response	<m:getResponse xmlns:m="http://example.org/gateway"> <rpc:result xmlns:rpc="http://www.w3.org/2001/12/soap-rpc"><ifSpeed>64000</ifSpeed></rpc:result> </m:setResponse>

The "getRequest" and the "setRequest" messages have a "version" element, which indicates the version of the SNMP, a "Community" element for authentication, and an "oid" element for object identification or a "path" element for addressing one or more object nodes in the DOM tree using the Xpath expression. A Query element is used to contain the XQuery expression for a complicated query. The "setRequest" element uses the "values" element to set a value of a node to be modified. There is "response " element for the "getRequest" and "setRequest", and the response element has "result" element as

the only sub element. The manager finds the appropriate method to invoke and pass the appropriate parameters to the method using the XML Schema. Table 2.10 presents a summary of the advantages and disadvantages of the three interaction translation methods. The DOM-based translation method is well suited for the internal gateway, interacting with a manager directly. The HTTP-based translation method provides an efficient and effective communication between the manager and the gateway, and reduces amount of request messages and data transfer. It is also easy to implement. The SOAP based approach has the advantage of the HTTP-based approach. In this approach the gateway can receive the request from and send the response to the XML-based manager in a standardized way and eliminates the pro

Table 2.10: Summary of different Translation Methods

Method	Advantages	Disadvantages
DOM-based Translation	No need to have a request handler a between gateway and a manager. It can be applied to both internal gateway and external gateway. Uses DOM as an intermediate storage for the manager.	Imposes a burden on the manager of invoking a series of interfaces for a request processing in appropriate order.
HTTP-based Translation	Easy to implement using the HTTP message extension. Provides an efficient mechanism for querying managed objects.	Need of Xpath/Xquery parsers
SOAP-based Translation	Simple to implement SOAP over HTTP. Inherits advantages of the HTTP-based translation. Provides a standard way to implement RPC.	Overhead of packaging SOAP messages.

CHAPTER 3

LITERATURE REVIEW

"XML-based Network Management, in which the structure of management information is defined using XML, the exchange of management data is in the form of an XML document, and it uses standard XML document processing methods to process the management data.."

J.P.Martin-Flatin [7] was the first person to propose using XML for network management in his research work on Web-based integrated network management architecture. He proposed two SNMP MIB to XML translation models.

- Model-level mapping: In this type of mapping there will be one DTD or XML Schema for each specific type of SNMP MIB object. Each element of the DTD or XML Scheme is represented to be the same as that of SNMP MIB variables or Object Identifiers. An example of model level mapping is listed below.

  <Interface>
  <Bandwidth type="string"> 100 Mbit/s </Bandwidth>
  <Interface>

  The advantage of the model level mapping is that the translated DTDs or XML Schema and XML document are easily readable for the users. This mapping is easy to parse and render graphically. The main disadvantage of the model level mapping is that it needs many DTDs or XML Schemas (i.e. one per SNMP MIB).

- Meta model-level mapping: There will be one generic DTD or XML Schema for all the SNMP MIB objects, that is there will be only one DTD or XML Schema per meta model. The XML elements have generic names such as *class, property, and operation.* These are the keywords defined for the meta model. An example of the meta model level mapping is shown below.

  <Class name=”interface”>
  <Property name=”bandwidth” type= “string”>
  <Value> 100 Mbit/s </Value>
  </Property>
  </Class>

  The main advantage of this mapping is its simplicity, that is one DTD or single XML Schema allows us to map all the SNMP MIBs. Its main disadvantage is that DTD are difficult to read, which makes debugging, and rendering more complex.

J.P. Martin-Flatin [7] presented an idea to use XML for integrated management in his research on web-based integrated network management architecture (WIMA) [5][22]. WIMA provides a way to exchange management information between a manager and an agent through HTTP. HTTP messages are structured with a multipurpose Internet mail extensions (MIME) multipart. Each MIME part can be an XML document, a binary file, BER-encoded SNMP data, etc. By separating the communication and information models, WIMA allows management applications to transfer SNMP, common information model (CIM), or other management data. A WIMA-based research prototype, implemented push-based network management using Java technology.

F. Strauss [23] [24] developed a library called “libsmi”, which can be used to access SMI MIB information. It can even translate SNMP MIB to other languages, like JAVA, C, XML, etc. This library has tools to check, analyze, dump, convert, and compare MIB definitions. The tool used for this called “smidump”.

Network devices developed by the Juniper Network are equipped with the JUNOS Operating system, which supports JUNOScript [25]. The JUNOSciprt allows the client applications to connect to the Juniper network devices and exchange messages as XML document. The request and response are represented as DTDs and XML Schemas. The communication between the client and network devices is through RPC requests. An XML-based RPC consists of a request and the corresponding response. It is transmitted through a connection-oriented session using any transport protocols like SSH, TELNET, SSL or a serial console connection.

Juniper network has already implemented a tool for mapping SNMP SMI information modules to the XML Schema. This tool is an extension of a previously implemented tool for converting SNMP SMI to Common Object Request Broker Architecture Interface Definition Language (CORBA-IDL). Currently Juniper network is working on implementation of XML document adapter for SNMP MIB modules using Net-SNMP and XML-RPC libraries.

In the 54th IETF meeting in July 2002 [22], a birds of a feather (BOF) session concerned with XML configuration (XMLCONF) was held. This BOF discussed the requirements for network configuration management and how the existing XML technologies, namely

SOAP, WBEM, SyncML, and JUNOScript could be used to meet those requirements. The Network Configuration (Netconf) Working Group was formed in May 2003. The Netconf Working Group is chartered to produce a protocol suitable for network configuration. The Netconf protocol uses XML for data encoding, because XML is a widely deployed standard that is supported by a large number of applications. XML also supports hierarchical data structures. The Netconf working group will take the XMLCONF configuration protocol as a starting point.

Web-based enterprise management (WBEM) [22] is an initiative of the DMTF and includes a set of technologies that enables the interoperable management of an enterprise. WBEM consists of a CIM, a DTD to represent CIM in XML, and a specification for CIM operations over HTTP. CIM provides a comprehensive object-oriented information model, and the CIM schemas are implemented not only for managing servers but also for network resources such as switches and routers. WBEM is currently being updated to include emerging standards such as SOAP. DMTF is representation of and the access to management data. DMTF is collaborating with OASIS to sponsor a new management protocol technical committee and to develop open industry standard management protocols.

The Alliance for Telecommunications Industry Solutions (ATIS) Technical Subcommittee [23] T1M1 (Internetwork Operations, Administration, Maintenance and Provisioning) is developing a Telecommunications Markup Language (tML) standard that would govern telecommunications network management. The tML is a language derived from XML and based on plain text tags that describe vocabulary used in the exchange of data between

telecommunications entities. The goal of the tML framework is to guide the development of interoperable operations, administration, maintenance, and provisioning (OAM&P) interfaces using XML for the telecom domain, to apply to various telecommunications OAM&P functions, and to provide a common framework in developing network management specifications by different groups. This recommendation is a framework containing rules, guidelines, and objectives for developing telecommunications industry standard tML schemas for OAM&P applications.

Jens Muller [23] implemented an SNMP/XML gateway as Java Servlet that allows fetching of XML documents on the fly through HTTP. MIB portions can be addressed through XPath expressions encoded in the URLs to be retrieved. The gateway works as follows. When an MIB module to be dumped is passed to mibdump, an SNMP session is initiated, and then sequences of SNMP GetNext operations are issued to retrieve all objects of the MIB from the agent. Mibdump collects the retrieved data and the contents of these data are dumped in the form of an appropriate XML document with respect to the predefined XML Schema.

Avaya [23] research lab developed an XML-based management interface to communicate with the SNMP enabled devices. They developed a tool for mapping SNMP MIB definition to XML Schema definitions.

Avaya research group developed a protocol using XML-RPC to retrieve and modify MIB information in SNMP enabled agents. In the mapping of the SNMP MIB to XML

Schema, most of the information that is not required is dropped from the XML Schema definitions.

Martin-Flatin proposed a way to convert the SNMP MIB to XML [7], but there is no algorithm for the conversion of SMI to XML. POSTECH developed an algorithm to translate the SMI to XML [6], and also developed three interaction translation methods.

Today's Network is equipped with legacy SNMP based agents, and it is difficult to manage legacy SNMP agents through an XML-based manager. Conversion of the XML-based request to an SNMP-based request through an XML/SNMP gateway provides the interaction between the XML-based manager and SNMP-based agents.]. For validation of the algorithm, POSTECH implemented an XML-based SNMP MIB browser using this

SNMP MIB to the XML translator. This gateway is developed by POSTECH at their DPNM laboratory [4] [6]. This gateway provides modules to manage networks equipped with SNMP agents [4]. The implementation of the gateway requires two types of translations: specification translations and interaction translations. The specification translation is concerned about the translation of the SNMP MIB to XML. POSTECH uses an automatic translation algorithm for SNMP MIB to XML. The interaction translation methods for XML/SNMP gateway are the process level interaction translation, the message level interaction translation, and the protocol level interaction translation.

The Network Management Research Group (NMRG) [23] of the Internet Research Task Force (IRTF) is a forum for researchers to discuss and develop new technologies to improve Internet management. In the year 2004, NMRG organized a meeting to

investigate the advantages and disadvantages of using web services technology for Internet management. In the meeting on web services, the participants discussed web services technologies, including SOAP, WSDL, and universal discovery description and integration, and compared them with SNMP. They also dealt with security in web services. NMRG's work in this area is in the early stage and has not yet produced any substantial results.

In our proposed work, we are implementing the manager and agent combination shown in [26] Figure 2.1(c), where we have XML-based manager communicating with SNMP agents via an XML/SNMP gateway. This paradigm uses HTTP as the communication protocol between the manager and the gateway, which is the interaction translation used is the same as that of the POSTECH. In our work, we address the limitations of the current XML-based network management. We provide a way to manage multiple network devices. We also provide a way to distribute the management work among multiple gateways thereby we will improve processing speed of the XML-based request. We also provide a mechanism for parallel processing of the XML-based request with in the gateway.

CHAPTER 4

FRAME WORK FOR EXTENSIONS TO XML-BASED NETWORK MANAGEMENT

4.1. MOTIVATION

The main drawback of the SNMP-based network management is the lack of scalability and inefficiency of processing the management data from the agents. We propose a framework to increase the efficiency of processing management data, decrease the communication cost and reduce the traffic between the XML-based manager and the XML/SNMP gateway. It takes advantage of the XML, DOM, and Java servlets.

An SNMP Get-Request operation gets the value of MIB objects from one agent at a time. If we want to get the same MIB value from *n* different agents then we need to execute the SNMP Get-Request operation *n* times. The SNMP Get-Bulk-Request operation can get the values of multiple MIB objects by traversing sequentially a MIB sub tree of one agent. In addition, SNMP Get-Bulk-Request allow to get bulk of data from one agent but does not allow to get the data from different agents in a single request. We propose a procedure to get data from multiple agents using single message. Similarly one can set the same MIB value to *n* different agents by means of single message.

In this framework, the XML-based manager can bundle one or more SNMP requests, which can be sent to one or more agents using a single message. This type of messages

will be useful when we want to issue the same request to many agents, or a Get-Request followed by a Set-Request to the same agent.

A manager may be required to get MIB objects from different agents that satisfy some conditions. For instance, when a manager is interested to get the same value from *n* different agents, it needs to execute *n* different SNMP get operations. This will increase the traffic between the XML-based network manager and the agents. With the XML-based network management, the gateway will check the conditions requested by the manager and sends back only relevant information.

A manager may also be required to set a MIB object after checking some conditions. In this case, it may need to first get the MIB value using an SNMP Get-Request operation then issue an SNMP Set-Request operation. We can define a single message that bundles multiple SNMP requests. This message will reduce the traffic between the XML-based manager and the gateway. This will increase the efficiency of the XML-based manager.

4.2. EXTENSIONS TO EXISTING XML-BASED NETWORK MANAGEMENT

In this section, we present the objectives of our work to the extensions to XML-based network management. The proposed extensions are described in the following subsections.

- Define a new message for a manager sending one request to multiple agents at the same time.
- Define a new message for a manager sending multiple requests to one agent.
- Define a new message for a manager sending multiple requests to multiple agents.
- Define syntax and translation scheme to support these new types of messages.
- Design a framework to enhance the existing system while still using legacy SNMP agents.
- Design and implement XML/SNMP Gateway for integration of SNMP and XML.
- Develop and implement the new framework and compare the results with existing systems.
- Performance Evaluation of the XML/SNMP Gateway.

4.2.1. Manager Sending One Request to Multiple Agents

We have designed two types of multiget operation, namely XML-based multihostget and XML-based multiobjectget. The general structure of these two XML-based multiget requests has been described in this section, and the following section will present an example.

- Multihostget: In this type of the multihostget operation, the values for the same MIB objects will be requested from all the agents. We have only one list of MIB objects for all the agents.

- Multiobjectget: In this type of multiobjectget operation, the values for different MIB objects will be requested from different agents. In this case, we have a different set of MIB objects for each host.

4.2.1.1. Multihostget

In this case, a request coming from the manager is addressed to multiple agents. The Java Servlet running at the server side receives the request. The servlet module creates the DOM tree representation of the multihostget request. The servlet module parses the XML request, and it takes the Xpath part of the request to extract the MIB nodes referenced. The hierarchical DOM tree representation of the manager sending one request to multiple agents through HTTP-based protocol is shown in Figure 4.1. The example of a manager sending one request to multiple agents is shown in Figure 4.2.

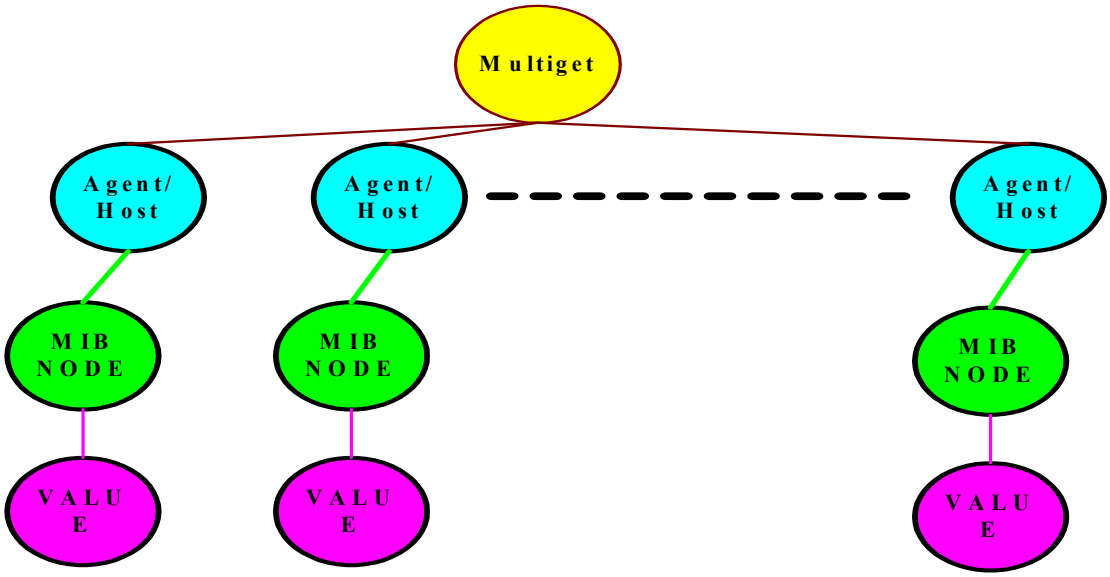

Figure 4.1: Hierarchical DOM tree of a Manager Sending a Single Request to Multiple Agents

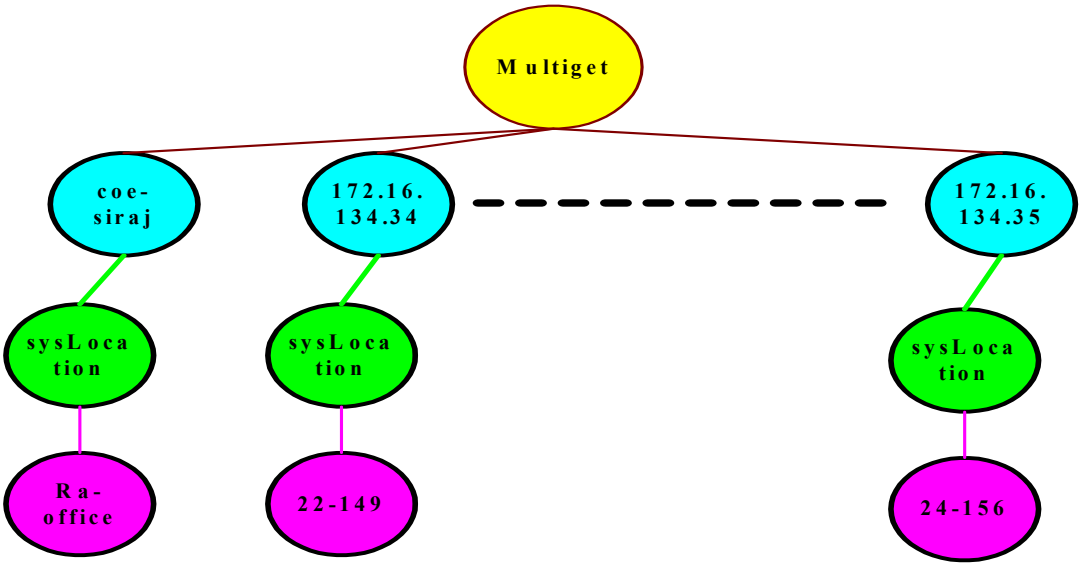

Figure 4.2: Example of a Manager Sending a Single Request to Multiple Agents

The general structure of the XML-based multihostget request sent by the XML-based manager to the gateway is shown in Table 4.1. The multihostget request has a required list of SNMP agent names, a list of MIB objects, the version, the read community, the write community, and the SNMP communication port for a group of agents. The hostlist contains a list of hostname tags. A hostname tag represents a target host name, which can be either the agent name or the IP address of that agent. The xpathlist contains a list of xpath tags. The value between these tags represents the target MIB object. The target MIB object can be either as scalar MIB object or a table MIB object or a column MIB object, or a group MIB object. The XML-based multihostget request resembles the SNMP Get-Request if we have only one agent in the hostlist. In this multihostget extension, we preserve the general structure of the SNMP Get-Request. Hence, the multihostget request can be used as the simple SNMP Get-Request with one host.

Table 4.1: General Structure of Multihostget Request.

General Structure of the XML-based Multihostget Request
<?xml version="1.0" ?> -<multihostget> - <Version>SNMPVersion</Version> - <WCommunity> Write Community String </WCommunity> - <RCommunity> Read Community String </RCommunity> - <Port> Port of SNMP Communication </Port> - <hostlist> <hostname> agent name or IP address </hostname> <hostname> agent name or IP address </hostname> …… <hostname> agent name or IP address </hostname> </hostlist> - <xpathlist> <xpath> MIB name </xpath> <xpath> MIB name </xpath> …… <xpath> MIB name </xpath> </xpathlist> </multihostget>

After Expansion of the multihostget request shown in Table 4.1, we get the XML-based multihostget request shown in Table 4.2 at the XML/SNMP gateway. A multihostget request has a list of child “host” tags and each host tag has a list of child “xpath” tags. All the host tags have an attribute “name” whose value represents the target agent. All the xpath tags contain an attribute named “MIB”, which represents the target MIB object. All the xpath tags have a “value” tag, which is used to store the value for the MIB object of this xpath tag. Initially, all the value tags have their value as “NONE”. After execution of the multihostget operation the value tag will be updated with the received response value. In the case of Table MIB objects the value tags are dynamically created according to the number of rows in the table object, which is we add a list of child value tags to the column MIB object. We will illustrate an example of a multihostget request in the following section.

Table 4.2: General Structure After Expansion of the Multi Get Host Request

General Structure of the Multihostget Request	
<?xml version="1.0" ?> -<multihostget> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> -<host name="agent name or IP address"> -<xpath MIB="MIB Object Name"> <value>NONE</value> </xpath> -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath> …… -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath></host> -<host name=" agent name or IP address "> -<xpath MIB=" MIB Object Name "> <value>NONE</value>	</xpath> -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath> …….. -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath> </host> …… …….. -<host name=" agent name or IP address "> -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath> -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath> ……. -<xpath MIB=" MIB Object Name "> <value>NONE</value> </xpath> </host></multihostget>

4.2.1.2. Example of Multihostget Request

An example of the XML-based multihostget request is shown in Table 4.3. The request includes two agents, and requesting two MIB objects, namely “sysContact” and “sysLocation”. This XML-based request is transmitted by the XMB to the gateway.

Table 4.3: Example of Multihostget Request

Example of Multihostget Request
<?xml version="1.0" ?> <multihostget> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <hostlist> <hostname>172.16.134.30</hostname> <hostname>coe-yousuf</hostname> </hostlist> - <xpathlist> <xpath>sysContact</xpath> <xpath>sysLocation</xpath> </xpathlist> </multihostget>

The XML-based multihostget request will be expanded for each agent and it looks as shown in Table 4.4. The host tags represent the target hosts, which are “172.16.134.30” and “coe-yousuf”. Each host has xpath child nodes that are used to represent the target MIB objects of the request, and which are “sysContact” and ”sysLocation” in this example. Xpath has a value tag that is initialized to a NONE value.

Table 4.4: Example of Multihostget request after expansion

Example After Expansion	
<?xml version="1.0" encoding="UTF-8" ?> <multihostget> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <host name="172.16.134.30"> - <xpath MIB="sysContact"> <value>NONE</value> </xpath> - <xpath MIB="sysLocation"> <value>NONE</value>	</xpath> </host> - <host name="coe-yousuf"> - <xpath MIB="sysContact"> <value>NONE</value> </xpath> - <xpath MIB="sysLocation"> <value>NONE</value> </xpath> </host> </multihostget>

Table 4.5 shows the final stage of the XML-based multihostget request after the received SNMP response values are updated. The SNMP MIB values are updated according to the agent name and MIB objects using XPath location expression.

Table 4.5: Example of Multi Get Host after updating with values

Example after Getting the values	
<?xml version="1.0" encoding="UTF-8" ?> <multihostget> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <host name="172.16.134.30"> - <xpath MIB="sysContact"> <value>siraj@ccse.kfupm.edu.sa</value> </xpath> - <xpath MIB="sysLocation.0"> <value>23-16B</value>	</xpath> </host> - <host name="coe-yousuf"> - <xpath MIB="sysContact.0"> <value>yousuf@ccse.kfupm.edu.sa</value> </xpath> - <xpath MIB="sysLocation"> <value>RA OFFICE</value> </xpath> </host> </multihostget>

Figure 4.3 shows the browser display for the XML-based response in HTML format. We applied an XSL style sheet to convert the result from XML to HTML.

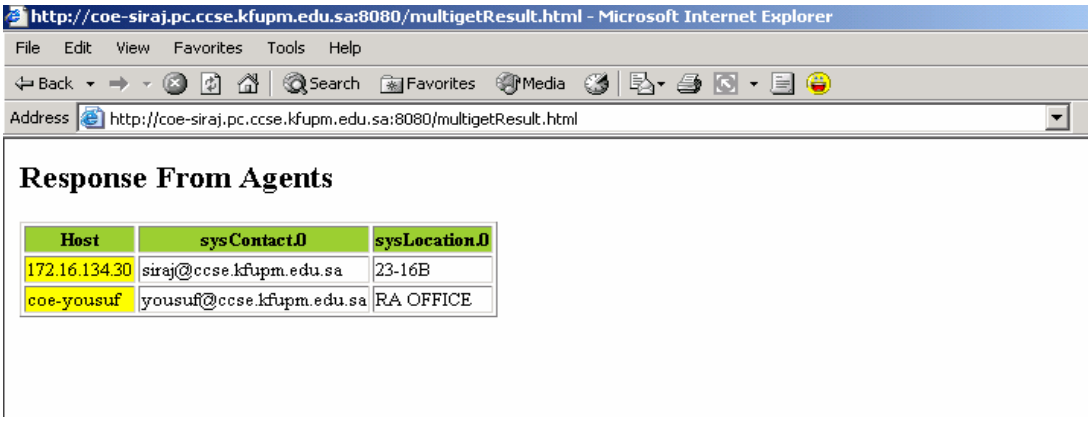

Figure 4.3: Response from Agents after applying transformation.

4.2.1.3. Multiobjectget

The general structure of the multiobjectget operation is given in Table 4.6 for multiple agents. Here for every agent we need to specify the required list of MIB objects, the version, the read community, the write community, and the SNMP communication port.

Table 4.6: General Structure of the Multiobjectget Request

General Structure of the XML-based Multiobjectget
<?xml version="1.0" ?> -<multiobjectget> - <host name=" agent name or IP address "> <Version>SNMPVersion</Version> <WCommunity> Write Community String </WCommunity> <RCommunity> Read Community String </RCommunity> <Port> Port of SNMP Communication </Port> <xpath MIB="MIB Object Name" > </xpath> <xpath MIB="MIB Object Name" > </xpath> …… <xpath MIB="MIB Object Name" > </xpath> </host> - <host name=" agent name or IP address" > …… </host> …… - <hostn name=" agent name or IP address" </host> - </multiobjectget>

4.2.1.4. Example of Multiobjectget

An example for multiobjectget request is shown in Table 4.7. It contains two agents. The manager is requesting different lists of MIB objects from the two agents. The first agent "172.16.134.30" is requesting three MIB objects whereas the other agent "coe-yousuf" is requesting two MIB objects. The multiobjectget request for each agent has separate tags for the version, the read community, the write community, and the SNMP communication port.

Table 4.7: Example of Multiobjectget Request

Example of Multiobjectget Request	
<?xml version="1.0" encoding="UTF-8" ?> - <multiobjectget> <host name="172.16.134.30"> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <xpath MIB="sysContact"> <value>NONE</value> </xpath> - <xpath MIB="sysLocation"> <value>NONE</value> </xpath> - <xpath MIB="sysName"> <value>NONE</value>	</xpath> </host> <host name="coe-yousuf"> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <xpath MIB="sysContact"> <value>NONE</value> </xpath> - <xpath MIB="sysLocation"> <value>NONE</value> </xpath> </host> </multiobjectget>

The servlet will first extract the agent list. Then using the name of the each agent, it will extract the list of MIB objects, the SNMP port, the SNMP version, the read community, and the write community from the request. Using this information, it will issue an SNMP

Get-request to every agent sequentially. The received response is updated. This is repeated for each agent. The final XML-based response is given in Table 4.8.

Table 4.8: Example of Multiobjectget after updating

Example after Getting the values	
<?xml version="1.0" encoding="UTF-8" ?> <multiobjectget> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <host name="172.16.134.30"> - <xpath MIB="sysContact"> <value>siraj@ccse.kfupm.edu.sa</value> </xpath> - <xpath MIB="sysLocation.0"> <value>23-16B</value> </xpath> - <xpath MIB="sysName"> <value>coe-siraj</value>	</xpath> </host> <host name="coe-yousuf"> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <xpath MIB="sysContact.0"> value>yousuf@ccse.kfupm.edu.sa</value> </xpath> - <xpath MIB="sysLocation"> <value>RA OFFICE</value> </xpath> </host> </multiobjectget>

4.2.2. Manager Sending Multiple Requests to One Agent

In this case, the XML-based manager sends one request, which consists of different SNMP operations to one agent. This request is passed to the XML request servlet, which parses the request and forwards it to the XPath/ XQuery module, where the Xpath and XQuery are separate. Then, a DOM tree is created as shown in Figure 4.4.

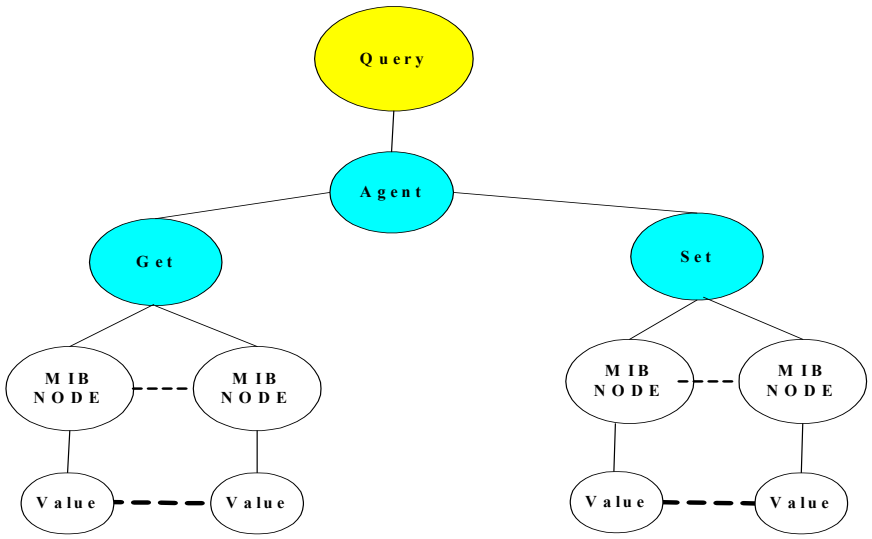

Figure 4.4: Hierarchical representation of the Multiple Request to one agent

Table 4.9: Example of Multiple Requests to one agent

Example of Multiple Requests to one Agent	
<?xml version="1.0" encoding="UTF-8" ?> - <multiple> <host name="172.16.134.30"> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <get> - <xpath MIB="sysDescr"> <value>NONE</value> </xpath> - <xpath MIB="sysName"> <value>NONE</value>	</xpath> </get> - <set> - <xpath MIB="sysContact"> <value> coe@ccse.kfupn.edu.sa</value> </xpath> - <xpath MIB="sysLocation"> <value>23-016B</value> </xpath> - </set> </host> </multiple>

An example of a manager sending multiple SNMP requests to one agent is given in Table 4.9. The example has one agent “172.16.134.30” requesting an SNMP Get-Request and an SNMP Set-Request. The XML-based request has a get tag and a set tag. The get tag has a list of XPath tags each representing a MIB object. Similarly, the set tag has a list of

XPath tags. The XPath tag of set has a value tag, which stores the value to be set for the MIB object.

4.2.3. Manager Sending Multiple Requests to Multiple Agents

Similarly in this case, the XML-based manager sends one request message, which consists of different SNMP operations to multiple agents. Figure 4.5 shows the general DOM tree of a manager sending multiple requests to multiple agents.

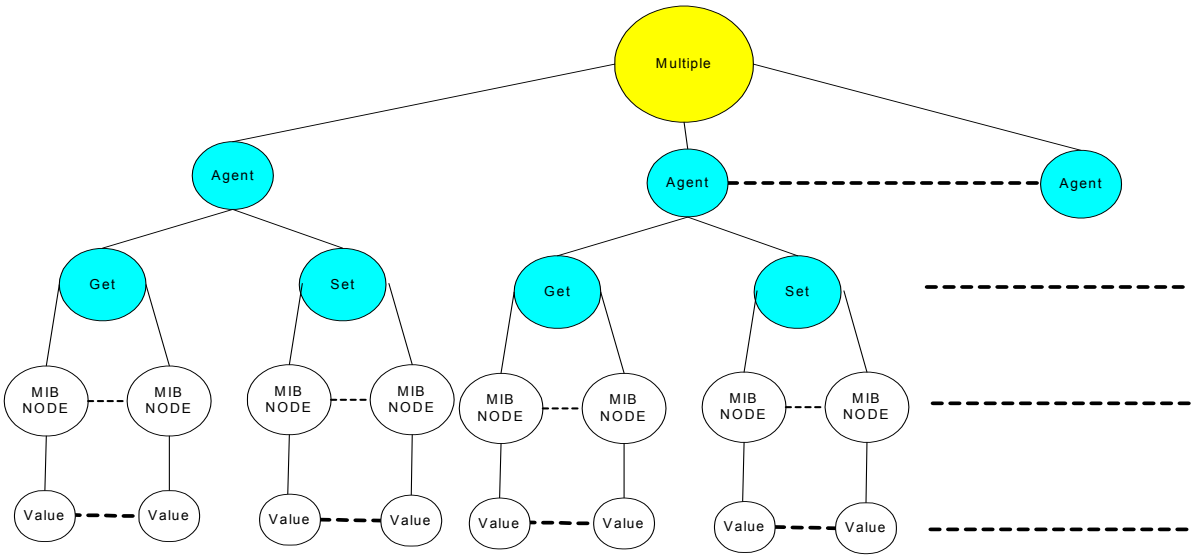

Figure 4.5: Hierarchical representation of the Manager sending Multiple Requests to Multiple Agents

An example of a manager sending multiple SNMP requests to multiple agents is given Table 4.10. The example has two agents “172.16.134.30”, and “172.16.134.230”. Both the agents are requesting an SNMP Get-Request and an SNMP Set-Request. The XML-

based request has a get tag and a set tag for every agent. The get tag has a list of XPath tags each representing a MIB object. Similarly, the set tag has a list of XPath tags. The XPath of the set has a value tag, which stores the value to be set for the MIB object.

Table 4.10: Example of Multiple Requests to Multiple agents

Example of Multiple Requests to Multiple Agents	
<?xml version="1.0" encoding="UTF-8" ?> - <multiple> <host name="172.16.134.30"> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <get> - <xpath MIB="sysDescr"> <value>NONE</value> </xpath> - <xpath MIB="sysName"> <value>NONE</value> </xpath> </get> - <set> - <xpath MIB="sysContact"> <value>NONE</value> </xpath> - <xpath MIB="sysLocation"> <value>NONE</value> </xpath> </set> </host>	<host name="172.16.134.230"> - <Version>1</Version> - <WCommunity> public </WCommunity> - <RCommunity> public </RCommunity> - <Port> 161 </Port> - <get> - <xpath MIB="sysDescr"> <value>NONE</value> </xpath> - <xpath MIB="sysName"> <value>NONE</value> </xpath> </get> - <set> - <xpath MIB="sysContact"> <value>NONE</value> </xpath> - <xpath MIB="sysLocation"> <value>NONE</value> </xpath> </set> </host> </multiple>

In the manager sending multiple requests to one agent and in the manager sending multiple requests to multiple agents, we first extract the agents present in the XML-based request and then for each agent we extract all the SNMP requests present in the XML-based request one after the other (get, set etc.). After extraction, the SNMP request will be executed and the response is updated with the received values.

4.3. OTHER POSSIBLE EXTENSIONS

In this section, we present some other possible extensions to the XNM that we have not implemented. These could be the subjects of some future work.

The communication between an XML-based manager and an RMON probe is similar to that of the communication between an XML-based manager and SNMP agents, since the RMON probe is going to be an agent for the top-level manager. The manager may request RMON probes to do the same type of monitoring.

The gateway is going to receive many alarms from the agents but the manager could request the gateway to send only the summary of the alarms by filtering the related alarms, or to send those alarms that satisfy certain conditions. Thus the gateway could act as a filter.

4.4. SOFTWARE REQUIREMENTS

Following are the required software to implement the extensions to the XML-based network management.

- **Java (JDK 1.4.3):** JDK 1.4.3 is Sun's software for developing java-based applications.
- **Apache Tomcat web server 5.0**: [27] Tomcat is the servlet container, which is used to run the Java Servlet and Java Server Pages. It is used in the gateway to receive the HTTP based request from the XML-based manager.

- **Xalan Xecers XML parser:** [28] It is an XML Parser, which supports DTD, Name Space, DOM API, SAX 2.0, JAXP 1.2, and XML Schema 1.0.
- **IReasoning SNMP API package:** [29] iReasoning Java SNMP API is the industry leading SNMP library, which provides a high performance, cross platform SNMP Java API for building network management applications. It is written in Java, and designed from the ground up to support fully all SNMP versions (SNMPv1, SNMPv2, and SNMPv3). All code bases are highly optimized to maximize performance and minimize overhead. This package is used in our system to implement the SNMP communication between gateways and SNMP agents.
- **SoftPerfect Protocol Analyzer:** [30] is an advanced, professional tool for analyzing, debugging, maintaining and monitoring local networks and Internet connections. It captures the data passing through a dial-up connection or a network Ethernet card, analyzes this data and then represents it in an easily readable form. This tool is used to capture the traffic between the XML-based manager and gateway, and between the gateway and SNMP agents. It is also used to find the response time between the transmission of an XML-based request and the reception of the corresponding XML-based response.
- **Web Browser (Internet Explorer 6.0):** It is a Microsoft product used to present the network management data in a user friendly format.
- **JPVM (Java Parallel Virtual Machine) source code:** [31] JPVM is a PVM-like library of object classes implemented in and for use with the Java programming

language. It is used to implement the JPVM master and slave gateways that communicate with SNMP agents, and for the distribution management tasks.

4.5. APPLICATIONS

In this section we give the functional area where the multiobjectget and multihostget are very useful such as configuration management and fault management. Configuration Management [7] [8] [32] is concerned about monitoring and controlling (i.e. get and set) parameters of managed devices. With the multi-get-request and multi-set-request we can get and set many objects on many agents using a single message. Thus, this proposed framework increases the efficiency of the processing, and thus the efficiency of the configuration management process. The new extensions can be used, for instance, to set an alarm threshold value in multiple agents or to find the location (i.e., sysLocation) of *n* agents by means of a single message. It will be also useful when the manager is interested in initializing many agents with the same value.

Fault Management [8] is concerned about detection and isolation of the problems that cause failures in the network. This gateway can be used to isolate minor and major alarms. The gateway can also be used to correlate different alarms and report to the manager a summary of the status of a sub network.

4.6. PROPOSED FRAMEWORKS

Our framework is based on the XML/SNMP gateway architecture, which was shown in Figure 2.1(c) [26], where communication is between an XML-based Manager, an XML/SNMP gateway, and SNMP agents. We propose three frameworks for the XML-based network management with XML/SNMP gateway.

- Single DOM Tree-based Approach.
- CSV-based Approach.
- JPVM-based Approach.

The functional description of these frameworks is presented in the following sections.

4.6.1. Single DOM Tree-based Approach

The proposed architecture for the single-DOM tree has three main components as shown in Figure 4.6:

- XML-based Network Management Station.
- XML/SNMP Gateway.
- SNMP agents.

The XML-based request is represented as an XML document. The XBM prepares and sends the XML-based request to the XML/SNMP gateway. The request is received by the XML request servlet, which retrieves the number of target agents present in the request. It extracts the Xpath component of the request and sends it to the Xpath/Xquery module,

which parses the XML-based request document. Parsing extracts the target MIB object present in the XML-based request received from the XBM.

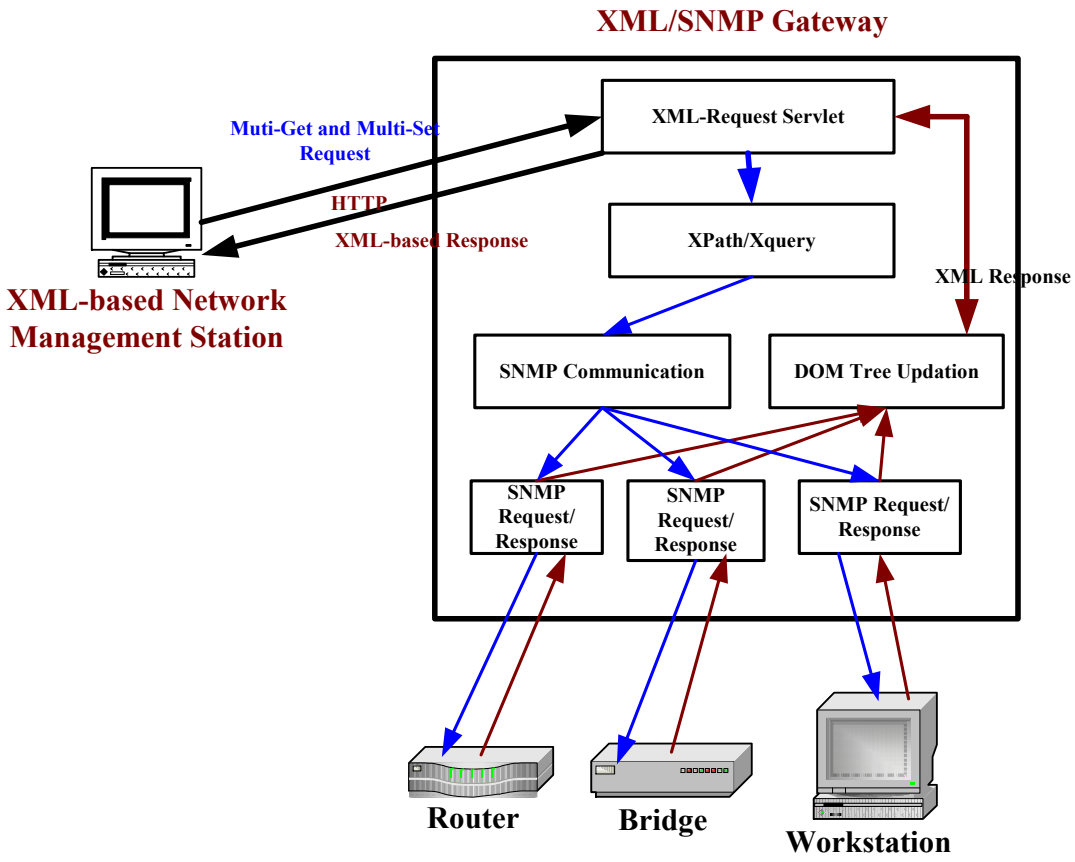

Figure 4.6: Single-DOM Tree based Framework

Using these target objects and the target hosts, the SNMP communication module will send the SNMP-based request to the agents and receives the SNMP response. The DOM tree is updated with the received response values. The updated response DOM tree can be translated into any form according to the user requirements using the XSL style sheets. Here in our approach we apply the XML style sheet to convert the response DOM tree into an HTML format and it is transmitted over the HTTP protocol to the XBM.

4.6.2. CSV-based Approach

The proposed architecture for the CSV-based approach is quite similar to that of the Single DOM Tree-based approach, and has the same three main components. The framework for CSV-based approach is shown in Figure 4.7. The CSV-based approach is different only at the updating of the SNMP response into an XML response. In CSV instead of updating the response to the DOM tree we write the response to a CSV file.

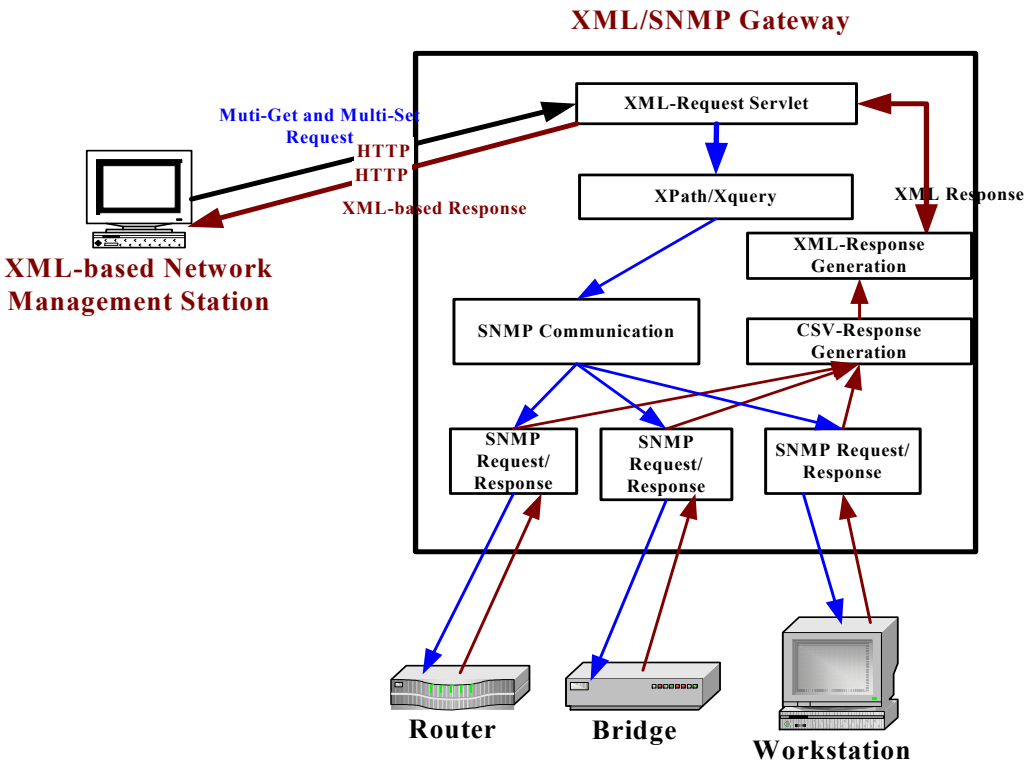

Figure 4.7: CSV-based Framework

The CSV response generation module handles the SNMP response received from the agents. Then, a CSV file for the received response values is created. Once the response is

received from all the agents the CSV file is converted into an XML document. The XML document can be translated into any form according to the user requirement using the XSL style sheets as described in the previous section

4.6.3. JPVM-based Approach

In this section we present the JPVM-based approach. First we give a general background of the JPVM, and then we describe the proposed architecture and its implementation. We also present the algorithms for load balancing and our contribution to JPVM.

4.6.3.1. JPVM Background

Adam J. Ferrari introduced JPVM [31] (Java Parallel Virtual Machine) library. The JPVM library is a software system for explicit message passing based on distributed memory MIMD parallel programming in Java. JPVM supports an interface similar to C and FORTRAN interfaces provided by the PVM (Parallel Virtual Machine) system. The JPVM system is easily accessible to the PVM programmers and has low investment target for migrating parallel applications to a Java platform. JPVM offers new features such as thread safety, and multiple communication end-points per task. JPVM has been implemented in Java and is highly portable among the platforms supporting any version of the Java Virtual Machine.

The JPVM system is quiet similar to that of a PVM system. JPVM has an added advantage of the Java as a language for network parallel processing. In the case of PVM,

we divide a task into a set of cooperative sequential tasks that are executed on collection of hosts. Similarly, in the case of JPVM, one has to code the implementation part into Java. The task creation and message passing is provided by means of JPVM.

4.6.3.2. JPVM Interface

In this section we explore the JPVM interface that provides the task creation, and execution. The most important interface of the JPVM package is the *jpvmEnvironment* class. The instance of this class is used to connect and interact with the JPVM systems and other tasks executing within the system.

An Object of this class represents the communication end-points within the system, and each communication point is identified by means of a unique *jpvmTaskId*. In PVM, each task has single a communication end-point (and a single task identifier), but JPVM allows programmer to maintain logically unlimited number of communication connections by allocating multiple instances of *jpvmEnvironment*.

First we need to set the JPVM environment on all the hosts that we are interested in parallel communication. For this, we need to run the *jpvmDaemon* java program on all the hosts. By running *jpvmDaemon* threads, we just initiate the JPVM environment. These threads are not used until all the hosts know about their JPVM environment.

Next we need to start the Console on one of the *jpvmDaemon* running hosts. The console program can be started running the *jpvmConsole* java program. Then, we have to register or add the other *jpvmDaemon* hosts to the host running the console program. We add the

hosts by giving the name and the port at which the *jpvmDaemon* started. This port is used during message passing between the JPVM hosts, and is the port through which the JPVM communication takes place.

4.6.3.3. JPVM Architecture

The proposed JPVM architecture is shown in Figure 4.8. It has mainly 3 components, namely an XML-based Manager, JPVM gateways, and SNMP agents. All the JPVM gateways are configured to run daemon processes. There will be one JPVM gateway that will run the *jpvmConsole* in order to notify all the hosts one another's existence and this is called the master JPVM gateway. The master JPVM gateway will communicate directly with the XML-based manager. The other JPVM gateways are known as slave JPVM gateways. These slave gateways communicate only with the master JPVM gateway. Hence, the JPVM-based network management is based on a master-slave paradigm.

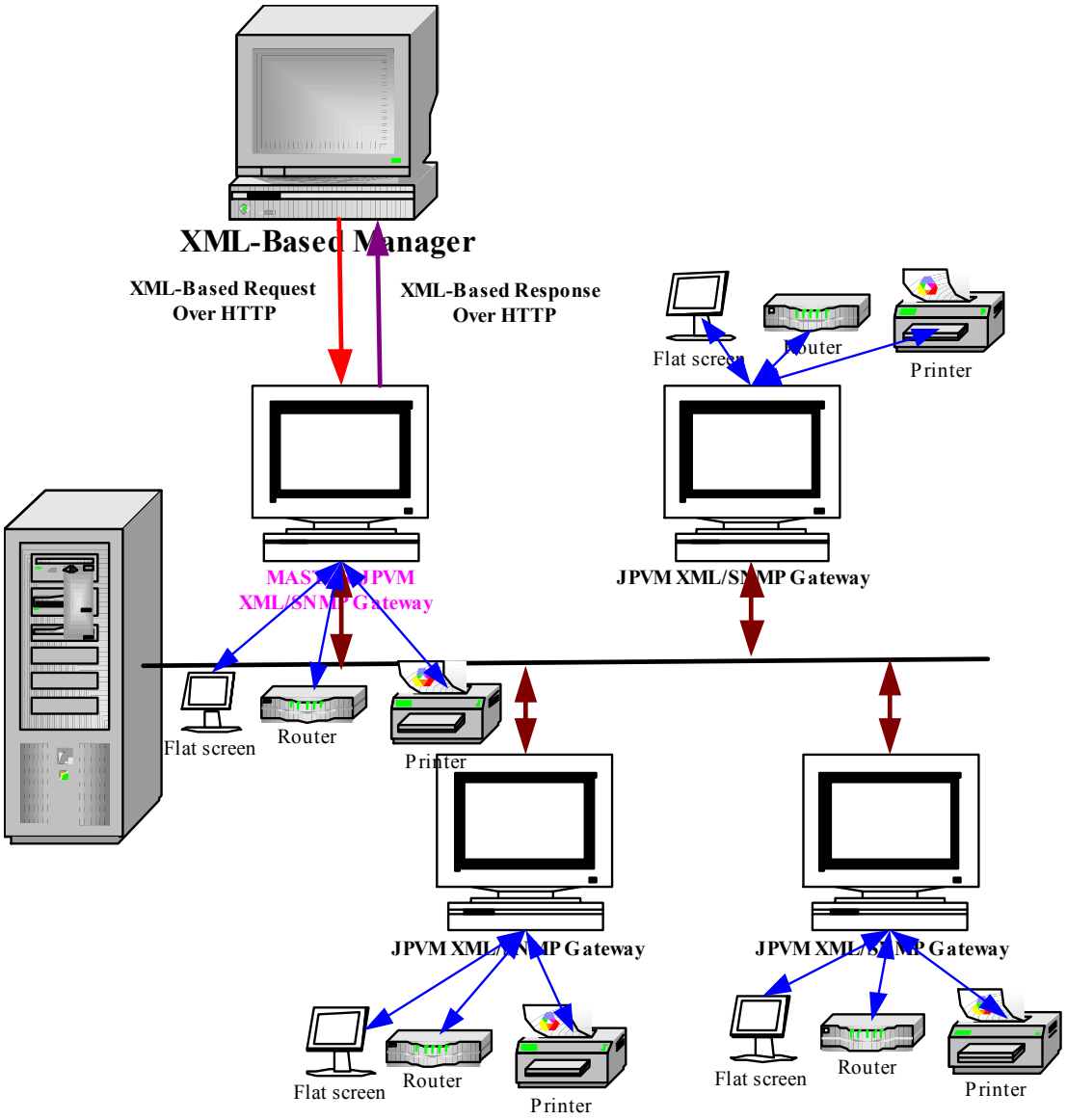

Figure 4.8: JPVM Framework for Parallel XML-based Netwrok Management

It has mainly 3 components, namely an XML-based Manager, JPVM gateways, and SNMP agents. All the JPVM gateways are configured to run daemon processes. There will be one JPVM gateway that will run the *jpvmConsole* in order to notify all the hosts

one another's existence and this is called the master JPVM gateway. The master JPVM gateway will communicate directly with the XML-based manager. The other JPVM gateways are known as slave JPVM gateways. These slave gateways communicate only with the master JPVM gateway. Hence, the JPVM-based network management is based on a master-slave paradigm.

4.6.3.4. Implementation of the Proposed Framework

The JPVM-based framework is implemented as a master-slave architecture, where a master JPVM is running at the web server since the XML-based request is send over HTTP protocol and is received at the web server. The master JPVM gateway receives the request from the XML-based manager. A *jpvmDaemon* program will be running on all the JPVM gateways. The master JPVM gateway is connected to a number of slave JPVM gateways, and will run the *jpvmconsole* program. The JPVM slave gateways have only the slave programs running on them for communication with the master JPVM and SNMP agents. The slave JPVM carries out the actual XML to SNMP translation and SNMP communication with the SNMP agents. The master JPVM status can be either working or not working. If the master has a working status, it can communicate with the SNMP agents after dividing the tasks since the master will be running separate *jpvmEnvirnment* task.

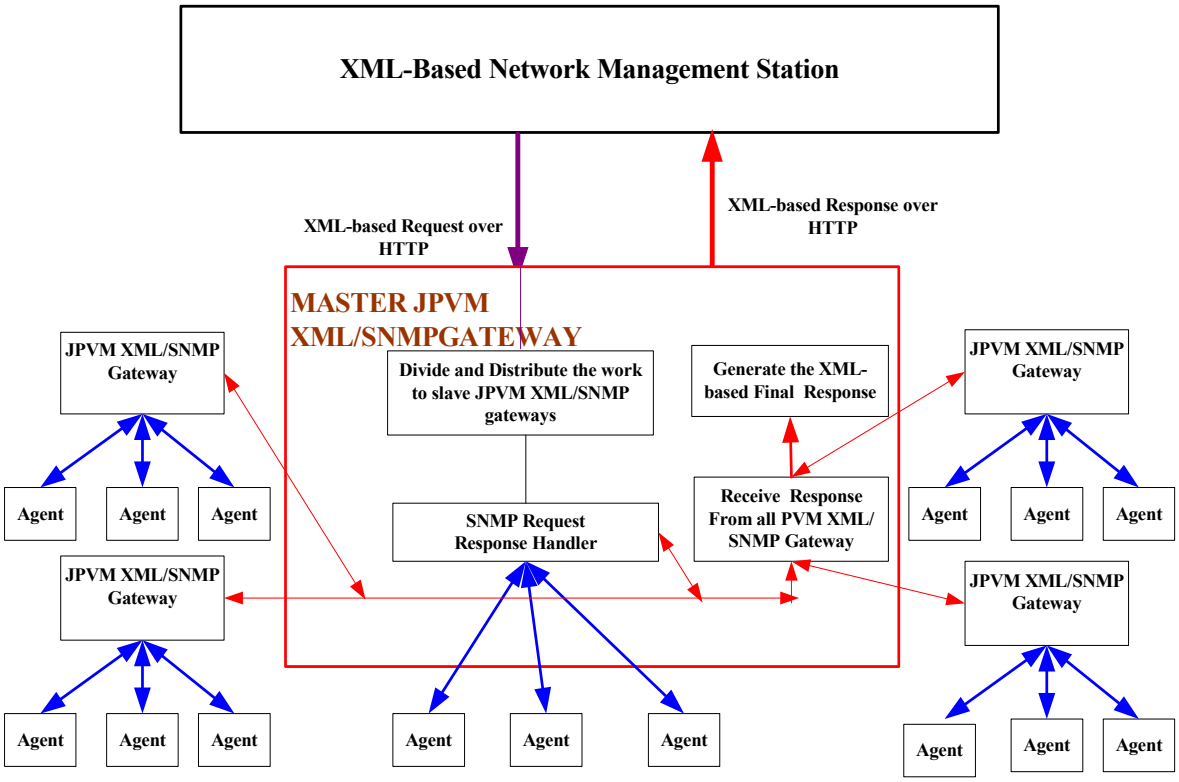

Figure 4.9: Implementation of the Proposed Framework

4.6.3.5. JPVM Master Algorithm

The JPVM master gateway algorithm is presented in Algorithm 4.1. The Master JPVM algorithm has three stages: initialization, waiting for the work, and termination. In the initialization stage, the master will start the JPVM environment, and create a pool of slave JPVM gateways and the character of the slave JPVM gateways is described in the next section. In the wait for request stage, the master will wait for the request from the XBM, and upon receiving the request it divides the work among the available pool of slave JPVM gateways, and dispatches the work to the slave JPVM gateways. It will wait for the

response from all the slave JPVM gateways, and after receiving all the responses, it joins them into one response document. Then, it will apply XSL to the XML document before transmitting the response over HTTP protocol to the XML-based manager. In the termination stage, the master JPVM will send the *stop* command to the slave JPVMs, and then exit from the JPVM environment.

Algorithm JPVM Master Gateway

```
Begin
        Initialization:
                Start the JPVM Environment
                Create Pool of JPVM Slave Gateways.
                Initialize the JPVM _Spawn for each Slave (Start of JPVM).
        Wait For Request:
                Divide the work.
                Send the work to each Slave JPVM gateways.
                Get the result from all the Slave JPVM gateways.
                Join the work.
        Termination:
                Send to each Slave the Stop command.
                Exit from the JPVM Environment.
End Master JPVM
```

Algorithm 4.1: Master JPVM Gateway Algorithm

4.6.3.6. Slave JPVM Algorithm

The slave JPVM algorithm is presented in Algorithm 4.2. The slave JPVM gateway starts the JPVM environment and parses the RFC-1213 MIB objects during the master JPVM initialization stage. The slave JPVM will wait for the work from the master JPVM gateway. Once the work is received from the master, each slave JPVM performs the Single DOM tree-based approach (i.e., Converting the XML-request into SNMP requests, sending SNMP requests, receiving the SNMP response, and updating SNMP responses in

the DOM tree). All the slave JPVM gateways will pass the XML response document to the master JPVM gateway. Then, all the slaves wait again for work from the master. This repeats until the master sends the terminate command to all the slave JPVM gateways.

Algorithm JPVM Slave Gateway

```
Begin
        Start the JPVM Environment.
        Parse the RFC-1213.
        While (true)
            Wait to receive the work from the Master.
              If (Stop)
                 Exit from the JPVM Environment
                   If (Work)
                      Get the XML-Document.
                      Do the Work.
                   Else
                      Break
         End While
         Exit from the JPVM Environment
End Slave
```

Algorithm 4.2: Slave JPVM Gateway Algorithm

4.6.3.7. Contributions to JPVM

JPVM supports basic data types like integer, long, string, character etc. The communication (message passing) between the different JPVMs is through these data types. XML-based network management requires communication by means of XML documents. The JPVM does not support message passing of XML documents among the different JPVM stations. In order to support message passing of XML documents, we added new data types such as: XML document, NodeList, Node, and SnmpPdu to the current JPVM source code.

4.6.3.8. JPVM Task Allocation

We classify the JPVM task allocation, based on the task or work from the master JPVM gateway to the slave JPVM gateways, into three types.

- Equal work to all slave JPVM Gateways.
- Weighted Static Load-Balancing.
- Dynamic Load-Balancing.

4.6.3.9. Equal work to all Slave JPVM Gateways

In equal work assignment, the master JPVM receives the XML-based request from the XML-based manager, and divides the request among slave JPVM gateways. Here the unit of work is the agent. If there are *N* slave JPVM gateways and the request contains *M* agents then the work for each slave JPVM gateway will be *M/N*.

Figure 4.10 shows the response time of two JPVM slaves, one with 350 MHz CPU, and the other with 711 MHz CPU. It can be seen that the same request is taking different times based on the processing capacity of the CPU. If we allocate the same amount of work to every processor then the high processing capacity processor will be underutilized. In order to maximize the utilization of the CPU processing capacity, we propose a weighted static load-balancing algorithm. The next section will illustrate this algorithm.

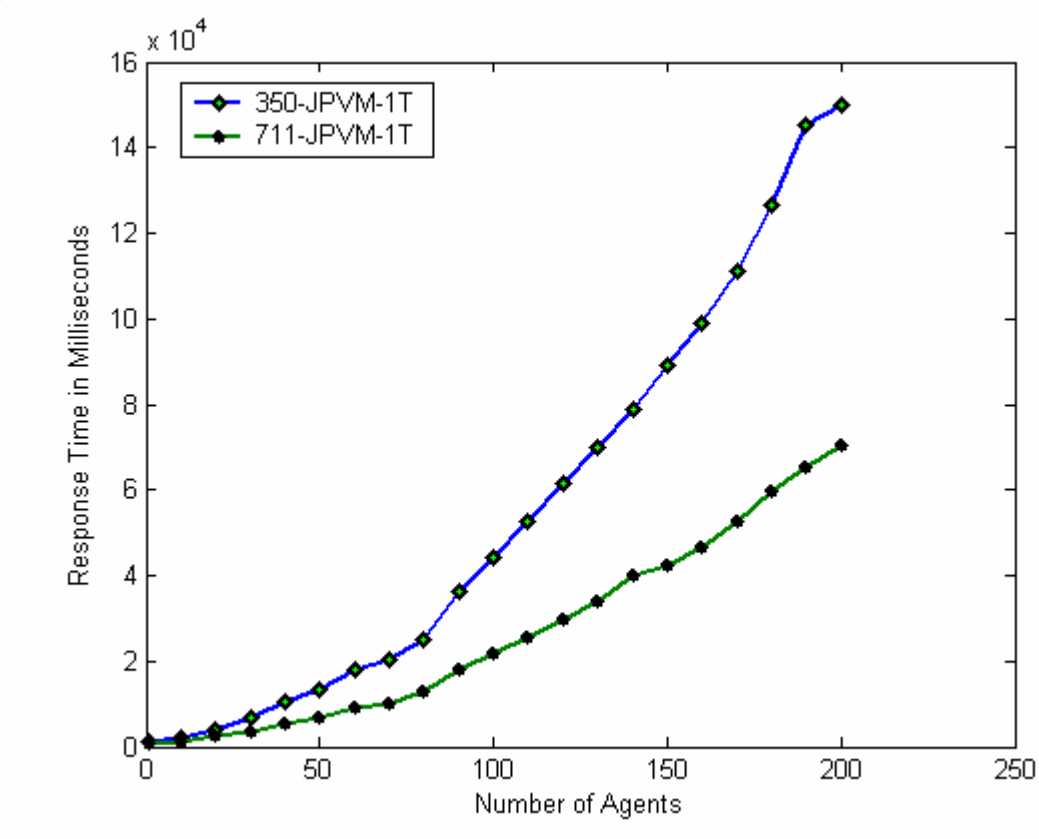

Figure 4.10: Response Time for JPVM Slave Running on different CPU speeds

4.6.3.10. Weighted Static Load Balancing

The equal work (i.e. dividing the work based on the number of slave JPVM gateways present in the pool) approach will provide good performance only for a homogeneous network of workstations. In the weighted static load-balancing algorithm, we divide the work based on the efficiency (processing speed of the workstations) of the workstations. This means that we assign a weight to the workstations depending on their processing speed, and during the work assignment it will be given work according to its weight.

The gateways may be busy serving some other requests. In such a case, efficiency of the weighted static load-balancing algorithm will decrease, i.e., the response time will increase. Instead of assigning the load based on a static weight, we assign the load based on the current load present on the slave JPVM workstation, which is dynamically assigning the load to the JPVM slave gateways. The next section will give brief background information on load balancing in general and our load-balancing algorithm.

4.6.3.11. Dynamic Load Balancing

In this section, we first give a brief introduction to the dynamic load balancing, and then we discuss our algorithm. Load balancing involves assignment of tasks to each processor in proportion to its performance. *The goal of load balancing is to assign a work proportional to the performance of the node or processor thereby minimizing the execution time of the application*. In ***Dynamic load balancing*** the assignment of tasks is done during runtime. The assignment of tasks is based on the current load on the processors (based on the performance of the processors).

Centralized Dynamic Load Balancing: In this type of dynamic load balancing there will be a centralized node, which is responsible for load balancing decisions. This centralized node will assign the tasks to all the other nodes (work is dispatched by the centralized master node). In our algorithm the master JPVM performs the assignment of work. Hence, our algorithm is based on the centralized dynamic load balancing paradigm.

The dynamic parallel algorithm [33] [34] divides the workspace into a *work pool* consisting of a large number of work blocks, each of which consists of a number of contiguous rows. Each row can be thought of as a sub task within the work block. In this dynamic load balancing approach, the master processor first creates the work blocks, and then distributes one work block to each of the slave processors. When a slave finishes a work block, it sends the computation back to the master and then the master sends the location of the next work block from the work pool for the slave to compute.

One benefit of work pooling is that, for a heterogeneous network, faster processors can request new work as soon as they are done without having to wait for slower processors. The dynamic parallel algorithm is tested with two heterogeneous systems, namely a 350 MHz and 711 MHz processing speed processors.

The Dynamic Load Balancing algorithm shown in Algorithm 4.3 has three stages:

1. Initialization: where the master JPVM creates a pool of slave JPVM gateways.
2. Wait for a request: where the master JPVM will wait to receive a request from the XBM, and then it will create a pool of working blocks from the request. The master processor then distributes one work block to each of the slave processors. When a slave JPVM finishes a work block, it sends the computation back to the master and then the master sends the next work block from the work pool for that XML/SNMP slave JPVM to compute. This repeats until all the blocks in the pool are completed. At the end, the master joins all the responses from the slave JPVM gateways.

3. Termination: where the master JPVM sends a STOP command to all slave JPVM gateways. This command tells the JPVM slaves to exit from the JPVM environment.

Dynamic Load Balancing-Master JPVM

```
Begin
        Initialization:
                Start the JPVM Environment
                Get Pool of JPVM Slave Gateways
                Initialize the JPVM _Spawn for each Slave (Start of JPVM)
        Wait For Request:
                Divide the work into Blocks
                Send the Initial work to each Slave JPVM gateway
                Do until Work Blocks Expires
                   Get the result From the Slave JPVM gateways
                   Send the work to Slave JPVM gateway
                End Do
                Join the work
        Termination:
                Send to Each Slave the Stop Command
                Exit from the JPVM Environment
End Master JPVM
```

Algorithm 4.3: Dynamic Load Balancing

4.7. IMPLEMENTATION WITH VARIATIONS

4.7.1. DOM Variations

The SNMP communication between the gateway and SNMP agents can be classified into two types, namely blocked and non-blocked. In the case of a blocked SNMP communication, the gateway sends a request to SNMP agents, and waits for a response. In

a non-blocked communication, the gateway does not wait for the response from the agent rather, it executes as a separate thread. The Single DOM Tree-based approach has been implemented in both a blocking and a non-blocking fashion. The SNMP responses received from the agents can be processed in three ways, sequential, producer–consumer, producer-consumer with message queue. The details of these methods are explained in the next subsections.

4.7.1.1. **Sequential Processing**

In this approach there will be only one thread running in the program. The program sequentially issues SNMP requests to agents one after another, and then processes the SNMP response from all the agents into an XML response. The sequential request and response processing is shown in Figure 4.11.

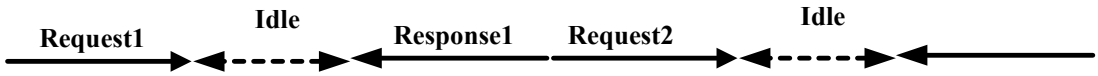

Figure 4.11: Sequential SNMP Request and Response

The sequential algorithm is shown in Algorithm 4.4 where we parse the request document, get the hostlist and MIB objects, and then communicate with the SNMP agents sequentially one after the other. Finally, the SNMP response is updated to the XML document to which the XSL is applied.

```
Algorithm Sequential
Begin
        Parse the XML-document
        Get the Hostlist
        Get the Target Objects
          For I=0 to length (Hostlist)
               Send SNMP-Request
               Receive SNMP-Response
               Update the Response on the DOM Tree
          End For
        Convert DOM tree to XML Document
        Send the Result Over HTTP after Applying
        XSL.
End –Sequential
```

Algorithm 4.4: Sequential Algorithm

4.7.1.2. Producer-Consumer Processing

The Producer-Consumer processing is a thread-based approach in which one thread produces (i.e., sends) SNMP requests and receives the values, while the second thread, (i.e., consumer) waits. Once the SNMP response is available from the producer, the consumer thread starts working on the received values. This continues for all the agents. In this approach, the producer thread must wait until the consumer processes the responses.

Algorithm 4.5 shows the producer consumer main algorithm with and without message queue, where we start the producer and consumer threads. Algorithm 4.6 shows the work for a producer and a consumer without message queue.

Algorithm Producer-Consumer	Algorithm Produce-Consumer with Message Queue
Begin Start the PRODUCER Start the CONSUMER Wait for Completion Convert DOM tree to XML document Send result Over HTTP after applying XSL End Prod-Con	Begin Initialize MQ Start the PRODUCER-MQ Start the CONSUMER-MQ Convert DOM tree to XML document Send result over HTTP after applying XSL End Prod-Con-MQ

Algorithm 4.5: Producer Consumer Algorithms with out and With Message Queue

Algorithm Producer	Algorithm Consumer

Begin
Parse the XML-document
Get the Hostlist
For I=1 to Length (Hostlist)
Send SNMP-Request
Receive SNMP-Response
Notify
Wait for Consumer
End for
End Producer
	Begin
	While (Producer has response)
	Receive the SNMP Response
	Update the DOM Tree
	Notify
	Wait for Producer
	End While
	End Consumer

Algorithm 4.6: Producer and Consumer Algorithm without Message Queue

4.7.1.3. Producer-Consumer with Message Queue

In this approach, there will be two threads similar to the producer- consumer processing. One thread will be working as a producer, and will get the values from the agents and the other will work as a consumer, and will process the values produced by the producer thread. In this approach, the producer thread does not wait until the received values are

processed. This approach employs one message queue where the produced values are stored. Whenever values are available in the queue the consumer thread processes them. The producer thread will be blocked when the queue is full. The consumer thread will be blocked when the queue is empty. The advantage of this approach is that the consumer thread is non-blocking when the producer thread is idle. And, the producer thread does not wait for the consumer thread to process the received values. Figure 4.12 shows the request/response of the SNMP operations. The producer thread has to wait for a response from the agent after issuing the request. There will be some idle time after issuing the request and before getting the response from the agent. This idle time is due to connection (session) establishment, data transmission, and network traffic.

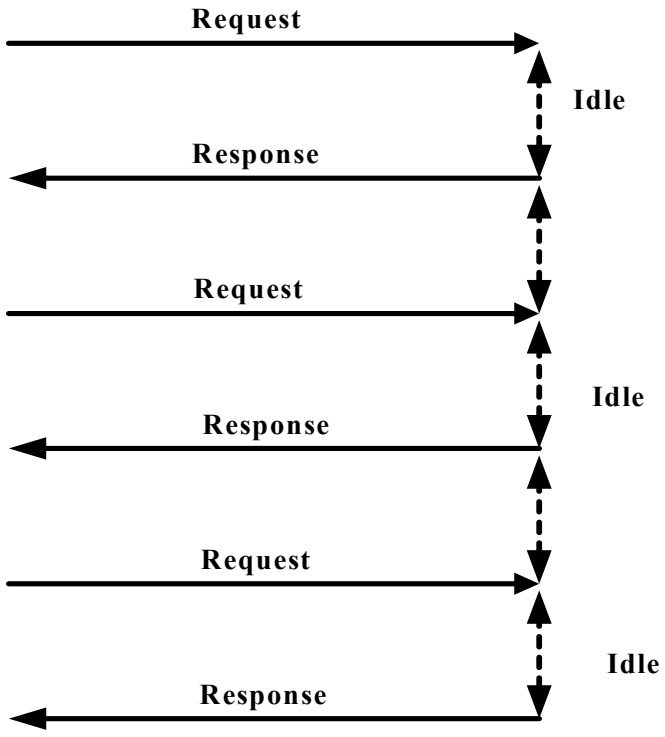

Figure 4.12: Request and Response of SNMP communication

Algorithm Producer with Message Queue	Algorithm Consumer with Message Queue

Begin
Parse the XML-document
Get the Hostlist
For I=1 to Length (Hostlist)
If (MQ is Not Full)
SNMP Request-Response.
Send the Response to MQ.
Else
Wait for MQ-Empty
End for
End Producer-MQ
	Begin
	Get the Hostlist
	For I=1 to Length (Hostlist)
	If (MQ is NOT Empty)
	Get SNMP Response from MQ
	Update the DOM Tree
	Else
	Wait for MQ-Not Empty
	End for
	End Consumer-MQ

Algorithm 4.7: Producer and Consumer Algorithms with Message Queue

The response times for the above methods are calculated but there is not much improvement compared to the sequential blocking method as the number of agents increases. The reason behind this behavior is that the paralliazation is only performed for the SNMP communication part which is only consuming a small amount of time compared to the XML to SNMP and SNMP to XML conversion. Hence, the result obtained did not show any improvement. There is no improvement in the response times whether the SNMP communication is blocking or non-blocking. The response times for these implementations is shown in Figure 4.13. The experiment is conducted for 100 runs on a Pentium IV process with 3.19 GHz CPU speed and 256 MB RAM.

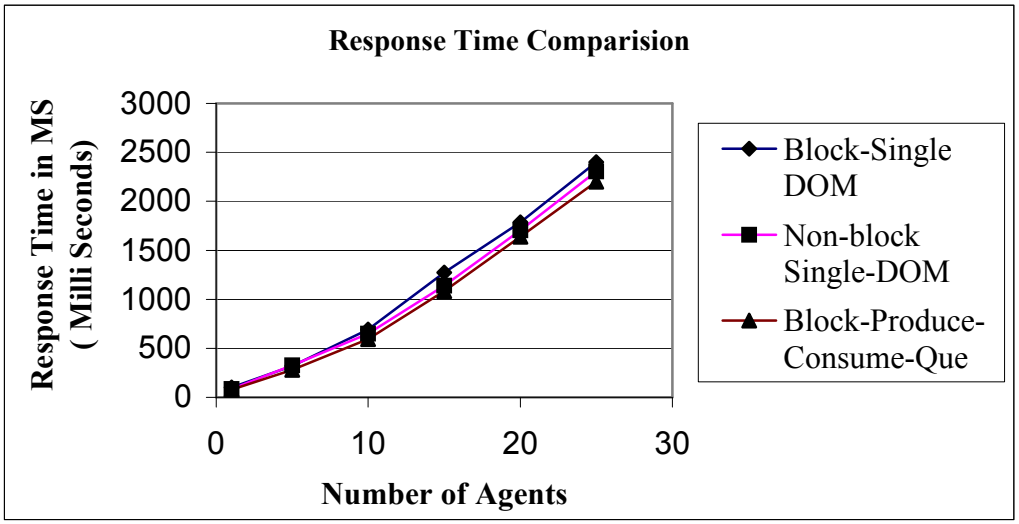

Figure 4.13: Response of Time of Single DOM with Blocking, Non-Blocking

In the next section, we describe ways to improve on the response time by parallelzing all the steps of an XML-based request

4.7.2. JPVM Variations

In one variation of the JPVM-based gateway implementation we got a high response time compared to the single DOM tree based implementation. In this implementation, we have the master JPVM running on the web server and the slaves running on other hosts. Whenever a request comes to the master JPVM, it will create (start) the slave JPVM gateways and then divide the work among them. This is repeated for every request received by the master JPVM. The creation of the slave JPVM gateways consumes lot of time due to the loading of the slave JPVM, creation of JPVM environment on the slaves, and parsing of the RFC-1213 for every request. This has been solved by loading the slave JPVM and creation of slave JPVM environment only once during first request from the

master JPVM. The slave JPVM will be executing continuously and waiting for work from master JPVM.

4.8. ADVANTAGES

Our proposed frameworks provide many advantages and are listed below.

- Configuration Management: The proposed extensions can be used for configuration management of multiple devices by sending a single request to multiple agents.
- Processing time: The processing time to process the XML-based requests has been reduced because of the distribution of management tasks among multiple slave JPVM gateways.
- Length of the requests: The proposed multihostget and multiobjectget will have shorter request message length than the POSTECH based get and set requests. It has been shown in Table 5.9.
- Access to multiple agents: It will provide a way to access multiple agents and send multiple requests in a single message. It has been shown in Figure 4.6, Figure 4.7, and Figure 4.8.
- Distribution: We can achieve a distribution of management tasks among the slave JPVM gateways. The slave JPVM gateways can then be assigned different management tasks. The quantitative results for distribution of tasks have been shown in Table 5.4 and Table 5.5.

- Parallelization: The same slave JPVM can be used to run many similar management tasks in parallel. The quantitative results for parallelization of tasks have been shown in Table 5.2

CHAPTER 5

PERFORMANCE EVALUATION AND COMPARISON

Our objective is to evaluate the effect of our approach on the scalability and efficiency of the NMS, and compare our results with the work performed by the POSTECH team. For this purpose, we will evaluate the response time, network traffic and message length of the multi requests.

Figure 5.1 shows the taxonomy of the frameworks used in the experimentation. The three approaches named single DOM, CSV, and JPVM are evaluated using both internal and external gateways. When the XML-based manager and the gateway are on the same machine, we refer to this as an internal gateway. And when the XML-based manager and the gateway are on two different machines, we refer to this as an external gateway.

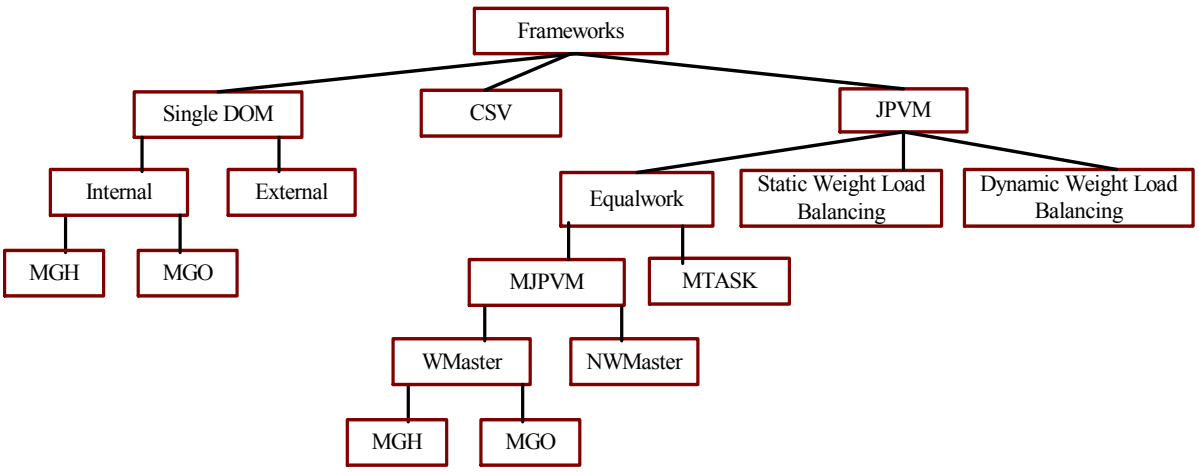

Figure 5.1: Frameworks for Experimentation

The single DOM and CSV based approaches are evaluated against internal and external gateways for both multihostget and multiobjectget XML-based requests. The JPVM-based approach is further classified into three methods based on the way the work is allocated to the slave JPVM gateways. The allocation can be either: equal work, static weighted load balancing, or dynamic weighted load balancing. These methods are evaluated for multiple slave JPVM gateways and multiple tasks running on a single slave JPVM gateway. These methods are also evaluated for a working master and a master JPVM gateway with no work. Two types of requests are used in the evaluation, namely multihostget and multiobjectget.

The next sections will present how we compute the response time, the network traffic, and the message length. Then, we will describe the experimental setup for the proposed extended XML-based network management.

5.1. RESPONSE TIME

The time elapsed between issuing the XML-based request from the XBM to the gateway and the time the response is received from the gateway back to the XBM is termed as the response time. Our objective is to compute the response time of the XML- based multirequest. The response time can be found by varying the following parameters:

1. The number of agents present in the multirequest.
2. The number of MIB objects present in the multirequest.

5.1.1. Response Time Calculation

The time elapsed between the issue of the XML-based request from the XML-based manager to the XML/SNMP gateway and the time the response is received from the XML/SNMP gateway back to the XML-based manager is termed as the response time. The Response Time between the XBM and SNMP agents is divided into five components as T1, T2, T3, T4, and T5 as shown in Figure 5.2.

- T1 is the time to send the XML-based request to XML/SNMP gateway over HTTP.
- T2 is the time required to convert (Translate) the XML-based request into SNMP based request. It includes building of the DOM tree for RFC-1213, and XML-based request.
- T3 is the time required to send the SNMP-based request to SNMP agents and get the SNMP based response from the agents. It is SNMP communication time.
- T4 is the time required to process the received SNMP response to XML-based response. It is the time required to convert the SNMP response to XML response.
- T5 is the time required to send the XML-based response to the XBM.

The response time components T2, T3, and T4 can be combined together and is named to be as SNMP-STACK communication. Finally we have the following components, transmission of XML-over HTTP to the gateway, SNMP-STACK communication, and

XML-transformation and transmission to the XML-based manger. We recorded the response total response time from using the SoftPerfect Protocol Analyzer from XML-based manager to the SNMP agents, and subtracted the response time form the XML/SNMP gateway to the SNMP agents to get the response time from XML-based manager to the XML/SNMP gateway.

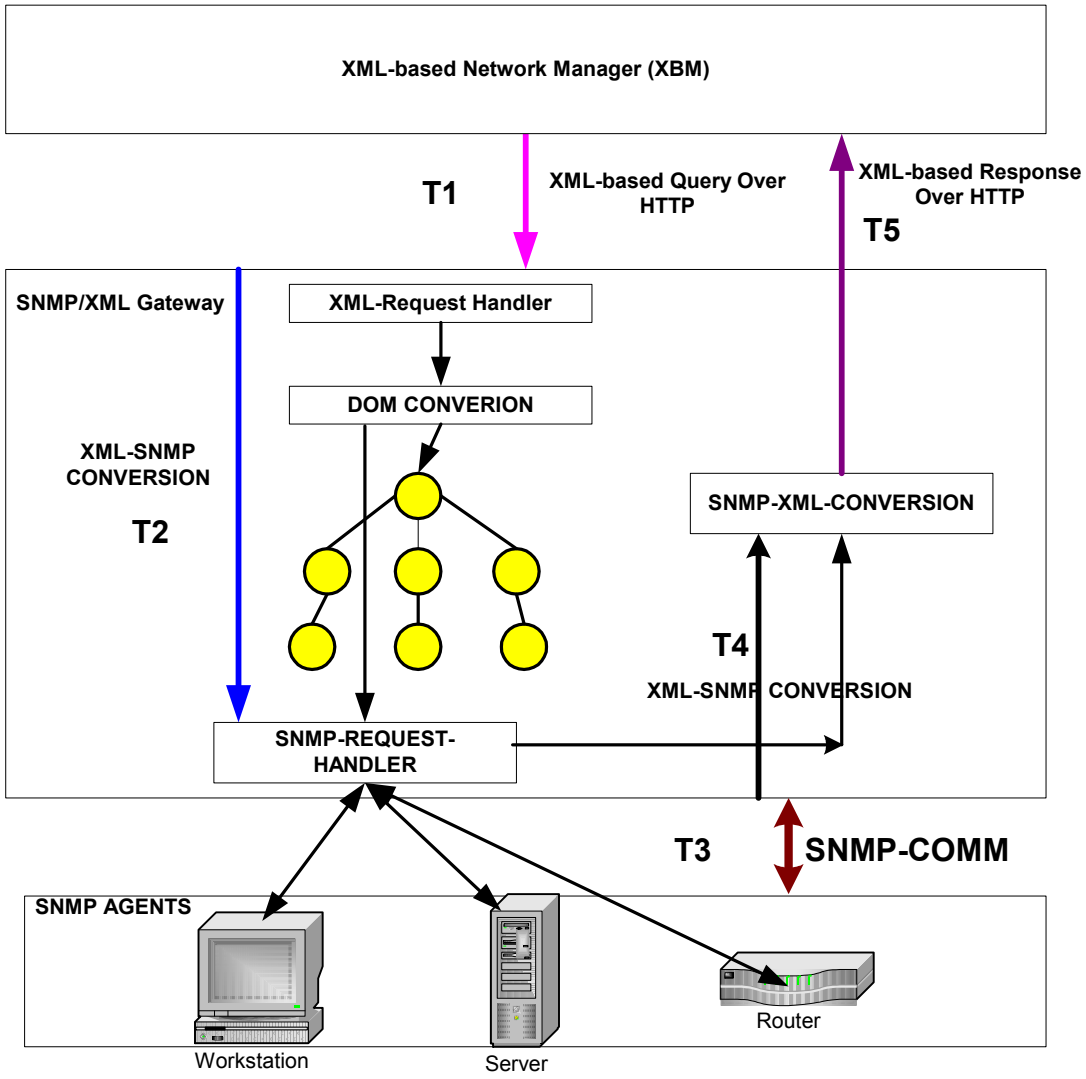

Figure 5.2: Response Time Calculation

5.2. EXPERIMENTAL SETUP

5.2.1. Experimental Setup-I

The experiment is conducted in our University campus and the experimental setup is shown in Figure 5.3. The XBM and XML/SNMP gateway are two PCs running Windows 2000. The XML/SNMP gateway has Apache TOMCAT 5.0 server running on it. The experiment is conducted inside the campus, and all the SNMP agents are connected over 100Mbps network connection that is connected over a Gigabit Ethernet backbone. The experiment is conducted for 25 runs. The maximum number of agents used in our experiment is 200.

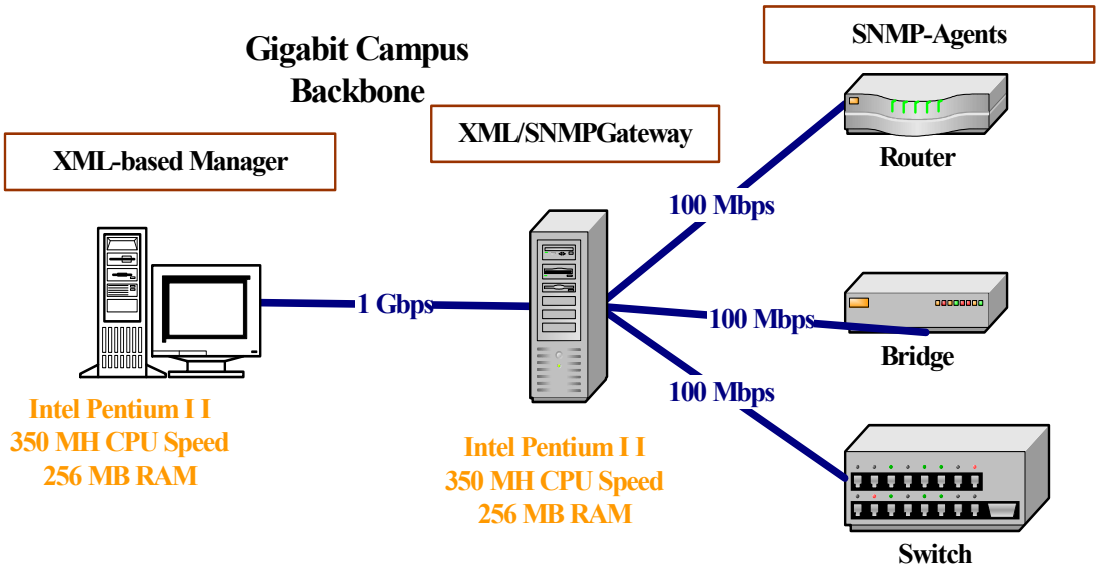

Figure 5.3: Experimental Setup-I

5.2.2. Experimental Setup-II

Figure 5.4 shows the experimental setup-II for JPVM-based network management. The master JVPM gateway is connected to a number of slave JVPM gateways. All the JPVM gateways are workstations running on Windows 2000 operating system. The master JPVM gateway has TOMCAT 5.0 web server running on it. The same experimental setup has been used with homogenous and heterogeneous systems. In the case of homogeneous systems, the slave JPVM gateways are of equal processing speed while in heterogeneous systems they are of different processing speed.

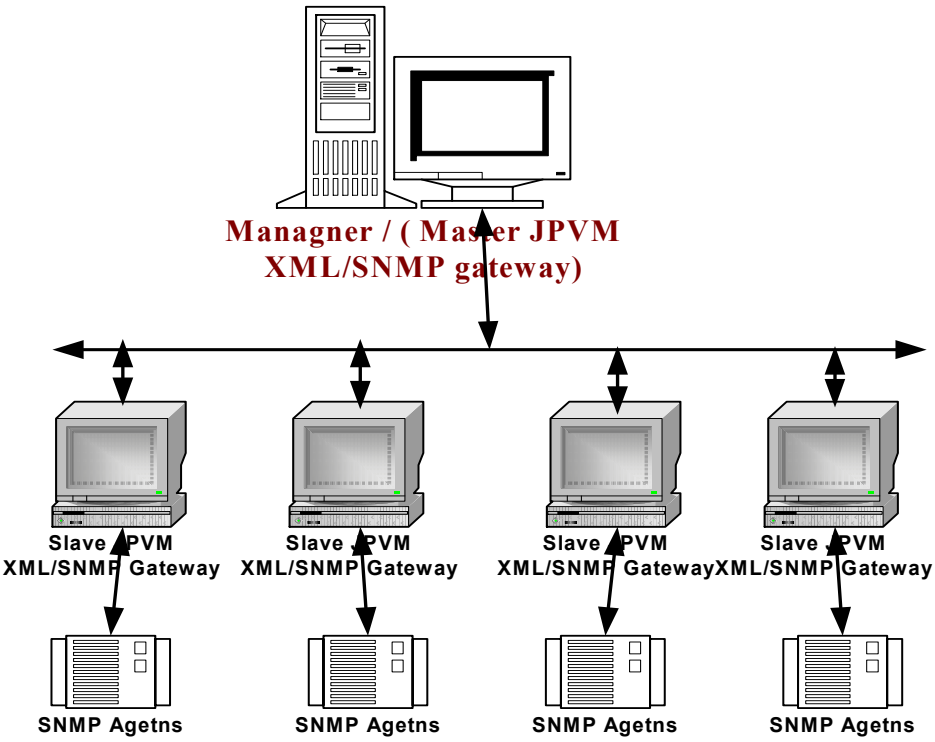

Figure 5.4: Experimental Setup-II

The experiment is conducted from our campus, and all the SNMP agents are connected over 100Mbps access network connection and a Gigabit Ethernet backbone. Each experiment was conducted for 25 runs.

5.3. EXPERIMENTAL RESULTS

5.3.1. DOM vs. CSV Results

Figure 5.5 shows the response time of the single DOM tree-based approach and CSV-based approach. The response is for system group MIB objects from RFC-1213. The CSV-based approach requires about half the response time compared to that of the single DOM tree-based approach.

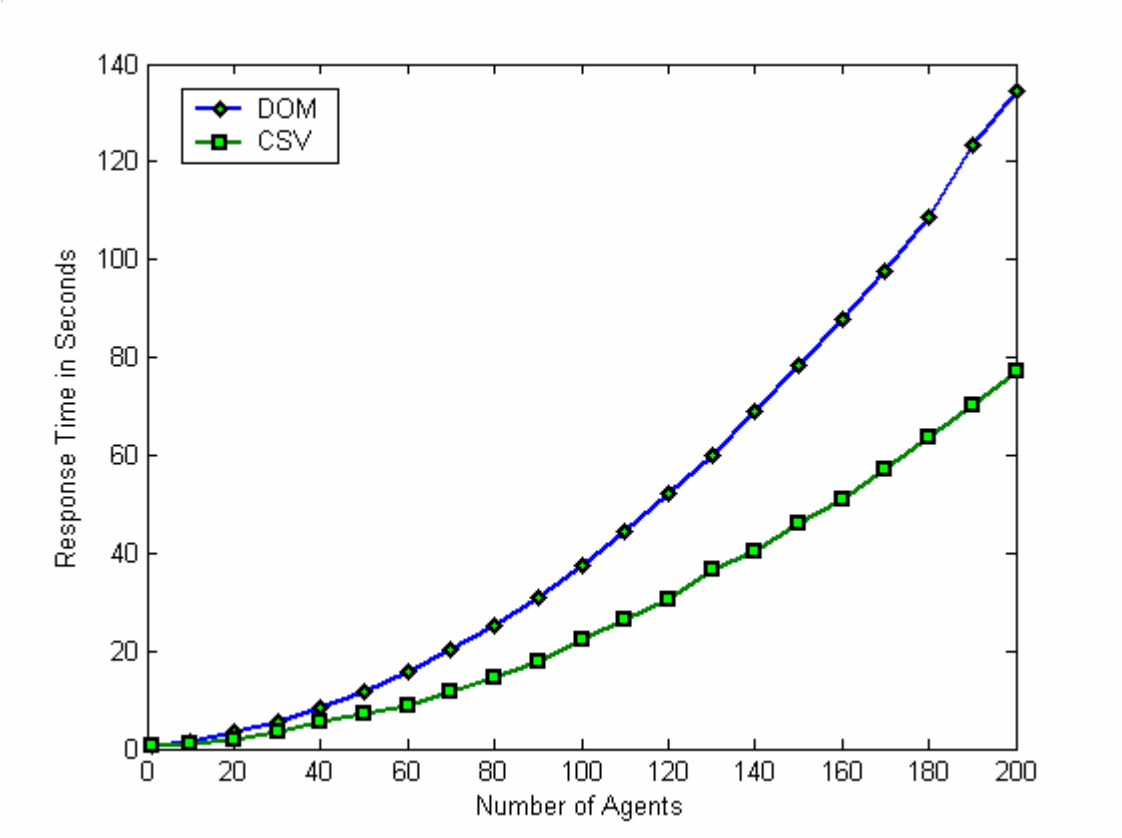

Figure 5.5: Response Time of DOM and CSV for System Group MIB objects.

The main reason behind the reduction in the processing time of the CSV based approach compared to the single DOM tree-based approach is that the DOM processing is used to build a single in memory object model of the XML-based request document. The advantage is that all the data can be accessed conveniently for whatever further processing requirement exists. The main disadvantages with the single DOM tree-based approach are:

- Time taken to process the whole model.
- Obvious resource problems when processing very large input files.

Figure 5.6 shows the various components present in the response time calculation of the single DOM tree-based approach. As the number of SNMP agents increases the SNMP communication component takes more percentage of time compared to the other components. Hence, most of the time is consumed during the SNMP communication between the gateway and the SNMP agents which includes the time for updating the DOM tree.

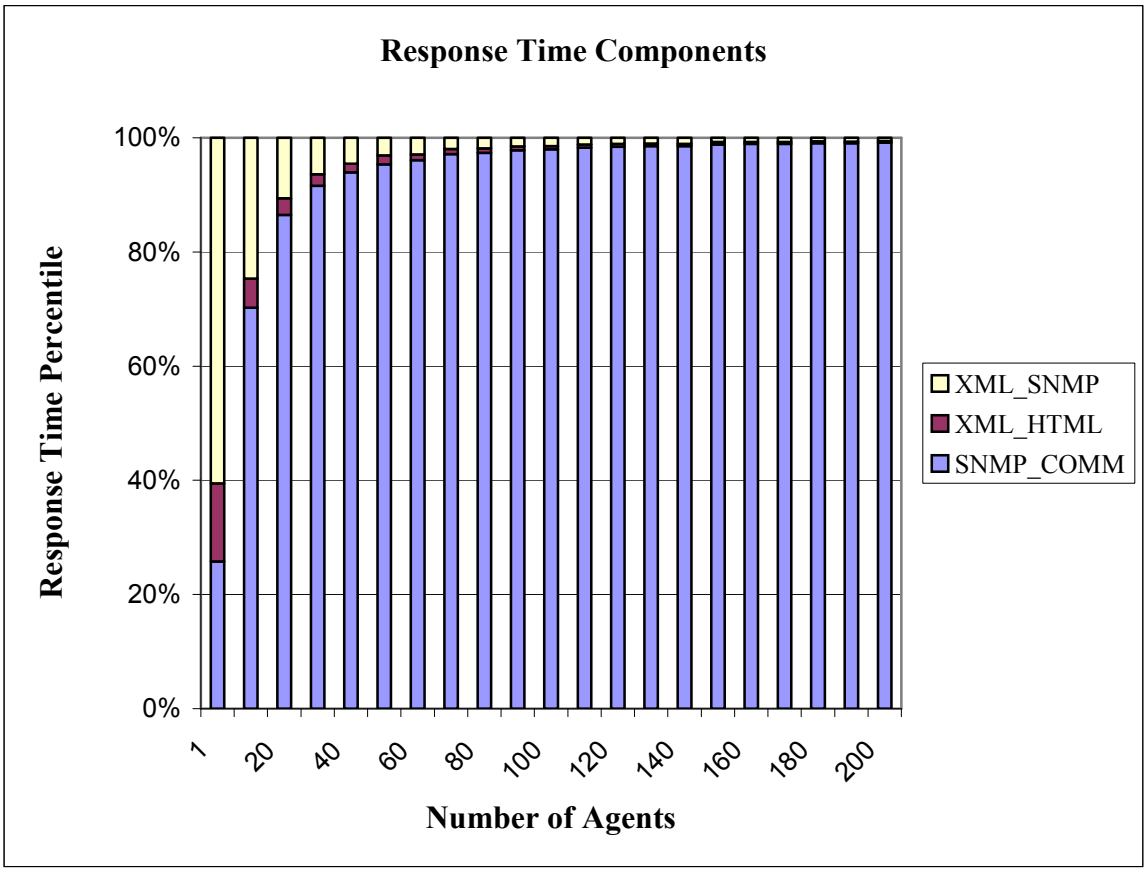

Figure 5.6: Various components present in the Response Time.

Table 5.1 shows the dissection of the single DOM tree-based approach and shows the response time at various stages. The first column shows the response time to communicate

with SNMP agents which is equal to the components of T3 shown in Figure 5.2 . This communication includes sending SNMP requests from the gateway to the SNMP agents, receiving the responses from all the agents and updating these responses into the XML-based response i.e.., updating the DOM tree after receiving the responses. The second column shows the response time required for transformation of the XML-based response into HTML and also the transmission time required to send the XML-based response to the XML-based Manager, which is equal to the components T4 and T5 shown in Figure 5.2.

Table 5.1 : Dissection of single DOM tree-based approach

	DOM	350	
No Agents	SNMP_COMM (T3)	XML_HTML (T4+T5)	XML_SNMP (T1+T2)
1	135	71.32	316.88
10	1073.48	78.04	377.04
30	5235.88	115.44	366.56
50	11133.24	188.32	357.24
70	19540.04	177.48	400.2
90	30352.88	210.36	469
100	36715.6	227.12	538.8
120	51218.92	262.44	541.88
140	68162.44	281.12	730.68
160	86826.52	316.04	622.48
180	107837.6	356.8	671.8
200	133477	375.4	725.1

The third column shows the response time required to translate the XML-based request into an SNMP based request, which is equal to the component of the T1 and T2 shown in Figure 5.2 . It also includes the transmission time required to send the XML-based request

to the gateway. Table 5.1 shows that the communication time is the main component that takes most of the response time.

Figure 5.7 presents the SNMP communication time of the single DOM tree based approach and the CSV-based approach. We can conclude that by employing the CSV in the gateway instead of updating directly the DOM tree we cut the response time to half. This is mainly due to the reduction of the SNMP communication time.

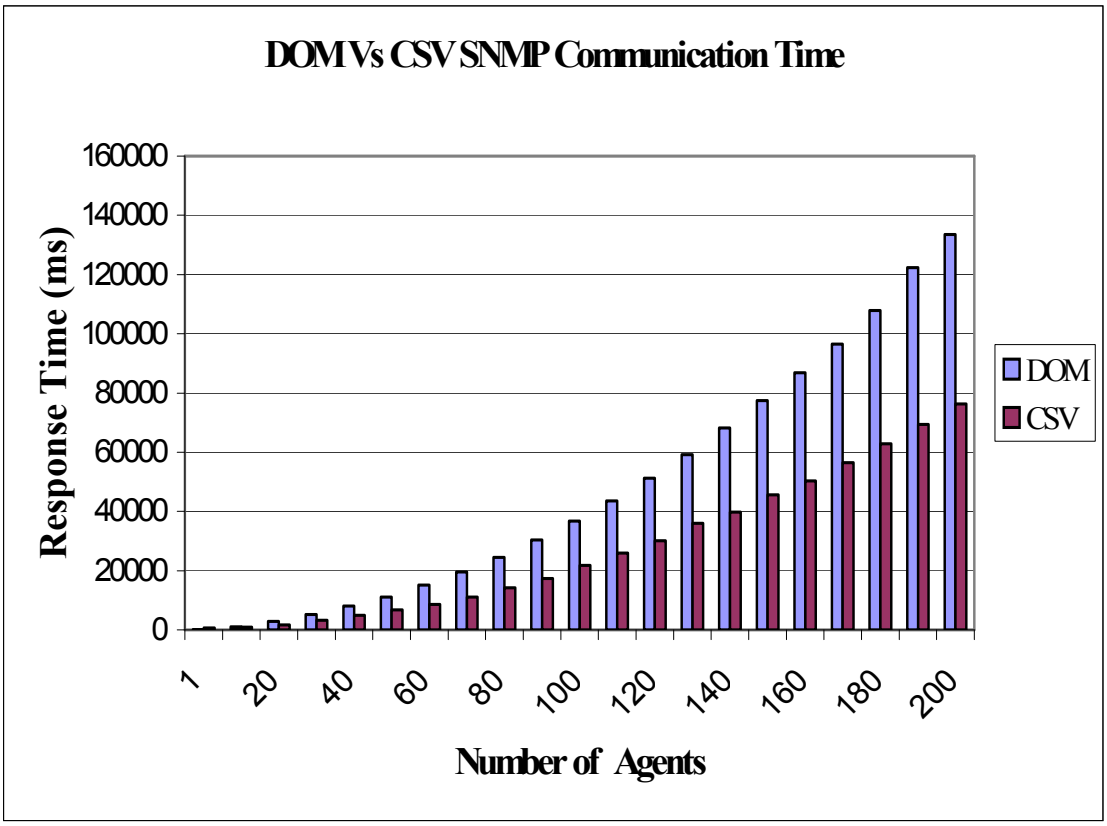

Figure 5.7: SNMP Communication component for DOM and CSV

5.3.2. JPVM-based Results

Figure 5.8 shows the response time for the JPVM-based approach with a working master and with a non-working master. The response time is shown for a single JPVM gateway with only one task running on the gateway. The JPVM with no work for the master has a slightly better response time compared to the working master gateway. The working master JPVM has 10 % higher response time compared to the non-working master JPVM. With a working master, the master gateway will be always busy and has to do more work compared to the slave JPVM gateways. The working master approach will not be a good approach as the number of slave JPVM gateways increases or when we adapt a hierarchical management of JPVM gateways. Since, in the case of a working master, as the number of slave JPVM gateways increases the work on the master JPVM increases due to its assigned load in addition to the processing of the response from all the other slave JPVM gateways. Hence, non-working mater JPVM will be suitable for hierarchical scalable network management paradigm.

Figure 5.9 shows the response time for the single DOM tree-based and JPVM-based approaches for the system group MIB objects with a single JPVM gateway and a varying number of JPVM tasks. The work assignment for all the tasks is equal. The results are for a 350 MHz processing speed processor with no work for the master JPVM gateway.

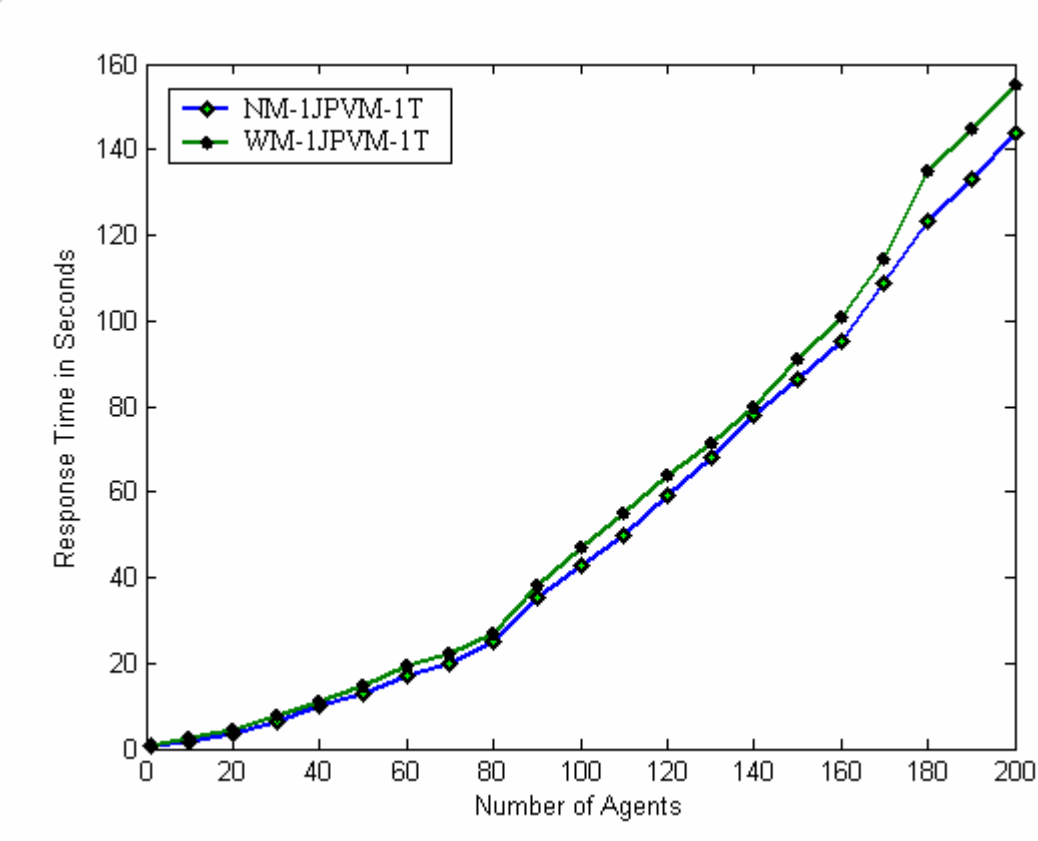

Figure 5.8: Response Time Comparison for System Group MIB Objects with JPVM.

The results are taken with 1, 2, 3, 4, and 5 parallel tasks running on the same JPVM gateway. We notice the time reduction of 40%, 57%, 64%, and 71% respectively for 2, 3, 4, and 5 tasks running on the JPVM gateway compared to the single task assignment. Hence, running parallel tasks on the single JPVM gateway will reduce the response time. The quantitative results have been shown for parallelization of tasks in Table 5.2.

Table 5.2: Quantitative Results for Parallelization of Tasks

Number of Parallel Tasks	Percentage of Reduction
2	40%
3	57%
4	64%
5	71%

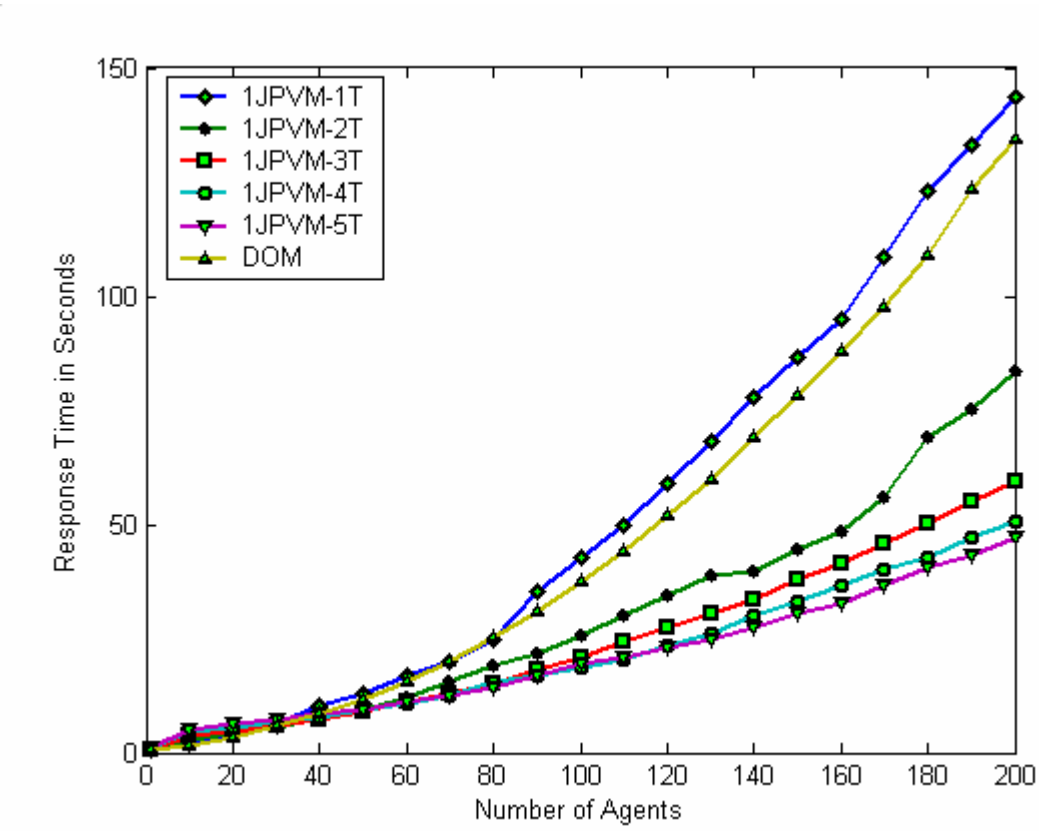

Figure 5.9: Response Time for DOM and JPVM with increasing number of Tasks with one slave JPVM.

Figure 5.10 shows the response time of the single DOM tree-based and the JPVM-based approaches with varying number of slave JPVM gateways, for the system group MIB objects. The division of work is equal among the slave JPVM gateways. The experiment is conducted on three slave JPVM gateways each with 350 MHz processing speed, and using 200 agents with no work for the master JPVM gateway. The response time of a single JPVM gateway compared with single DOM tree-based gateway is 5% higher due to the extra time for task creation. The reduction in the time with two JPVM gateways and 200 agents will be equal to the time that a single JPVM takes with 100 agents. Similarly, with three slave JPVM gateways, this time will be equal to the time for 67 agents running on the single JPVM gateway. We notice 71% and 85% reduction of time with two and three slave JPVM gateways respectively compared to the single JPVM gateway.

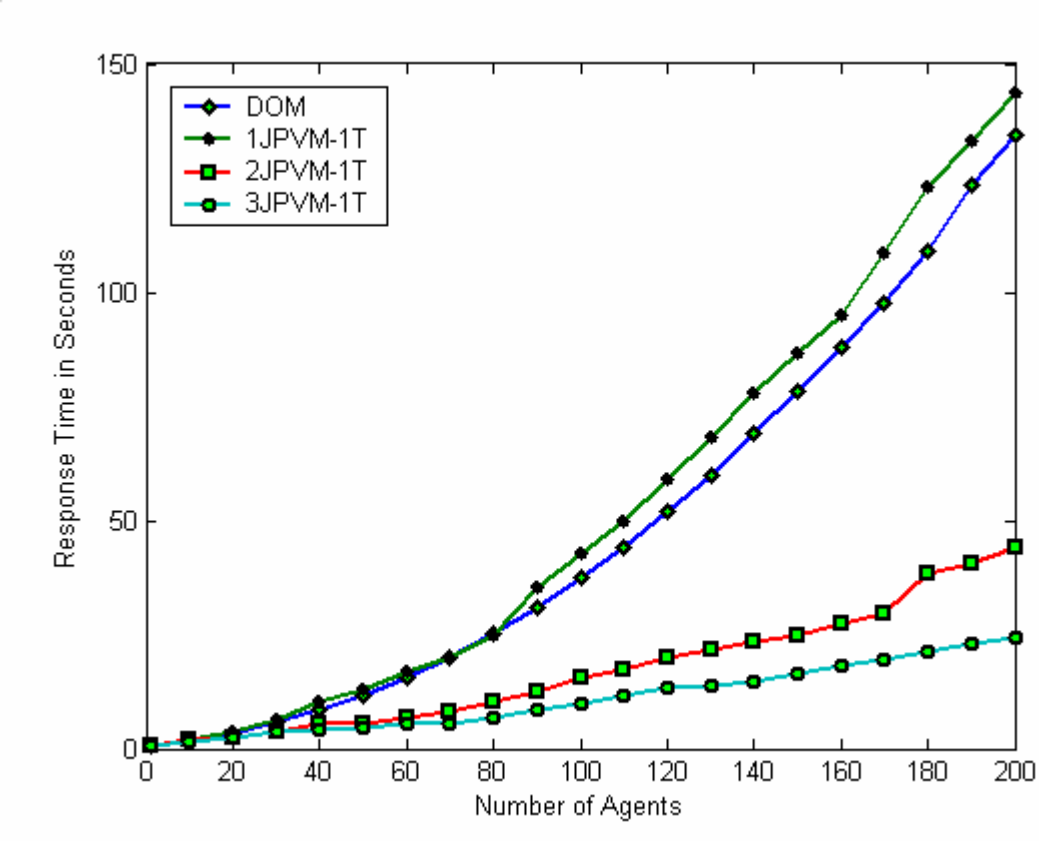

Figure 5.10: Response Time for DOM and JPVM with varying slave JPVM gateways.

Figure 5.11 shows the response time for system group MIB objects for the JPVM-based approach with two slave JPVM gateways compared to a single JPVM gateway with increasing number of tasks. The increase in the number of slave JPVM gateways from one to two has shown significant reduction in response time compared to the increase of the number of tasks running on a single slave JPVM gateway. The response time with two JPVM gateways is better than that obtained with 2, 3, and 4 tasks running on a single slave JPVM gateway. Hence, the increase of the number of parallel slave JPVM gateways

will provide a better performance than the increase in number of parallel tasks on a single slave JPVM gateway.

Table 5.3 presents the total response time for the system group MIB objects as the number of JPVM gateways increases. The experiment is conducted with four JPVM gateways, one of which is the working master JPVM gateway. The third column shows the total response time with two JPVM gateways, and the response time for 200 agents is 45645.4 milliseconds, which is approximately equal to the time for a single JPVM with 100 agents (46917.4 milliseconds).

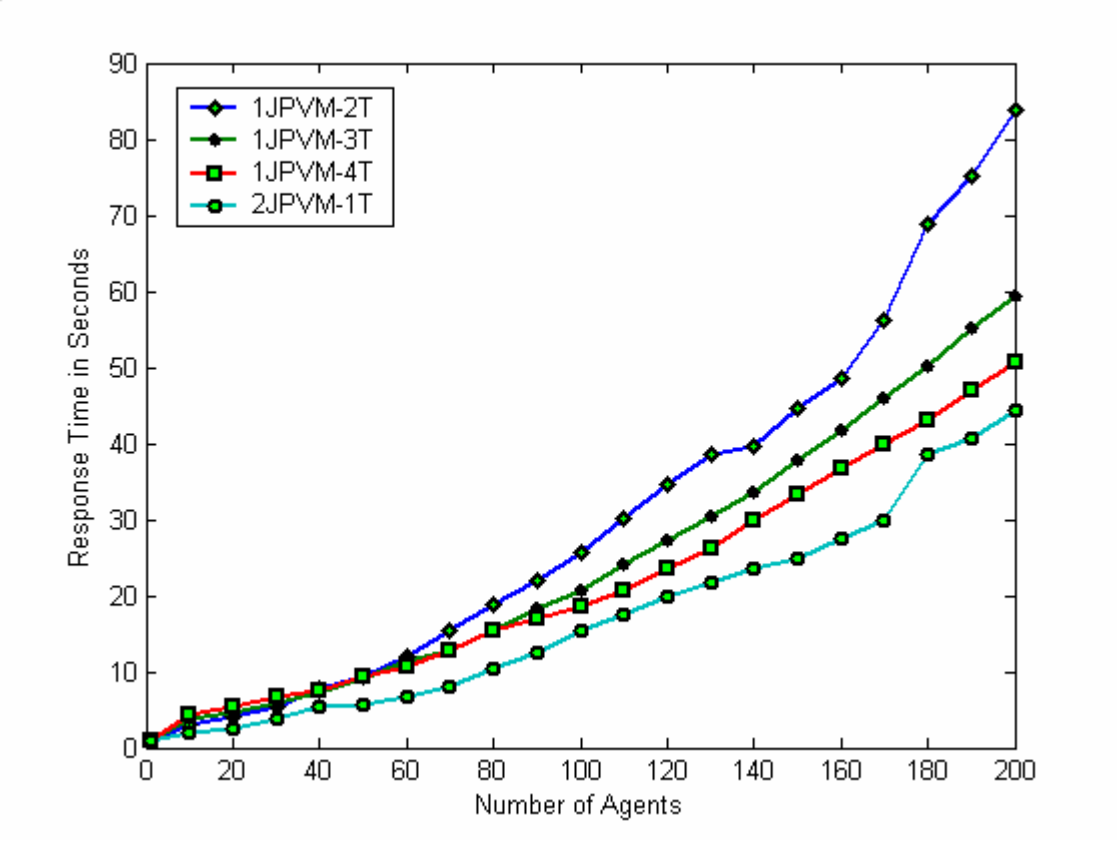

Figure 5.11: Response Time for JPVM with increasing Tasks on single slave JPVM and with two slave JPVM.

Similarly, with 3 and 4 JPVM gateways, it will be equal to the time for running on a single JPVM with 67 and 50 agents, respectively, in addition to the communication and processing cost as the number of JPVM gateways increases. Figure 5.12 shows the graph for the working master with a varying number of tasks running on it for the values in Table 5.3.

Table 5.3: Response Time for single JPVM with increasing number of tasks with a working master gateway in milliseconds.

		Gateways		
Agents	**JPVM-1**	**JPVM-2**	**JPVM-3**	**JPVM-4**
1	717.2	863.4	763	793.2
20	4542.2	2613.8	2441.4	2389.4
40	11116	5483.8	4158	3637.4
50	14609	5892.4	4915	3707.2
60	19446	7149.6	5780.2	4240
70	22187.8	8874.6	5896.6	5472
80	26816.6	10835.6	6493.4	5686.2
100	**46917.4**	16942.4	10393	8606.4
180	135076	39871.2	21551	15899
190	144984.6	42148.6	23449.8	16285.4
200	154966.8	**45645.4**	**25214.2**	**17873.8**

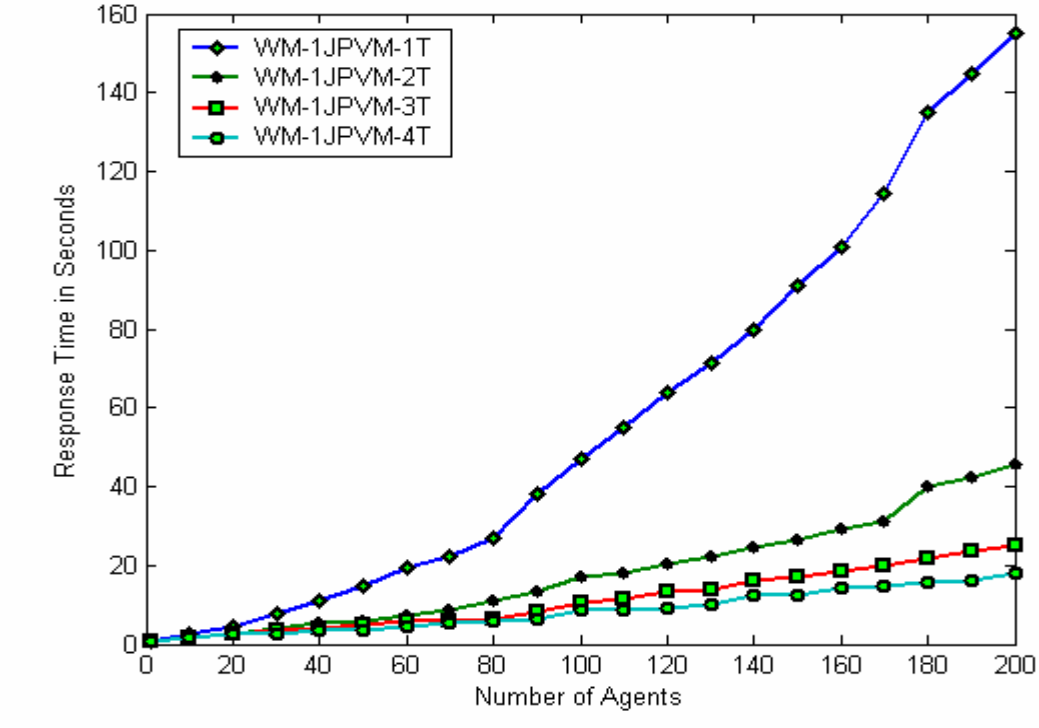

Figure 5.12: Response Time for a Working Master with Varying JPVM.

Figure 5.13 shows the response time for our XML-based multiobjectget compared with the POSTECH XML-based request. The response time in our experiment is 16% more compared to the POSTECH's, because our experiment is conducted on a 711 MHz processing speed Intel Pentium III processor whereas the POSTECH conducted the experiment on the 800 MHz processing speed Intel Pentium III processor. However, we have shown that with the CSV-based & JPVM-based approach we obtain better results than the basic DOM-based approach. Thus, our frameworks provide better results than POSTECH's.

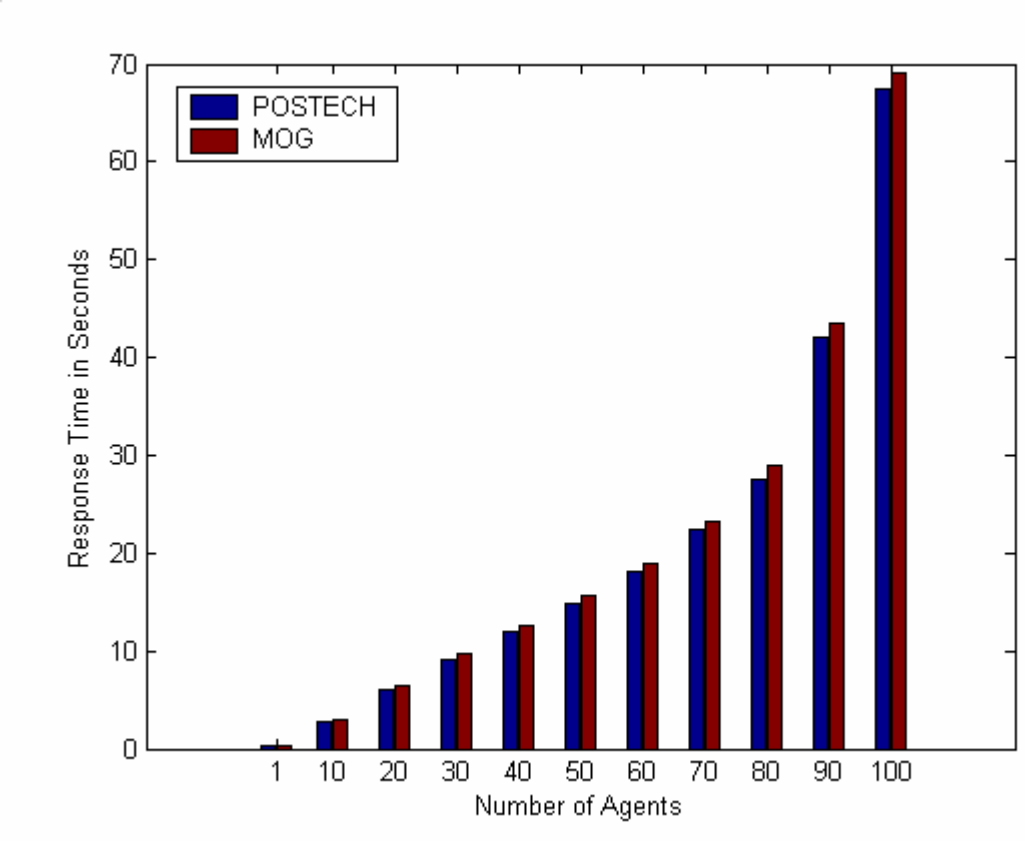

Figure 5.13: Response Time POSTECH compared with Multiget Objects.

Figure 5.14 shows the response time with varying number of MIB objects present in the XML-based request. The response time is recorded for one agent varying the number of MIB objects. The response time increases by ~ 500 milliseconds for every 10 additional MIB objects. The response time increases linearly in function of the number of MIB objects in the request.

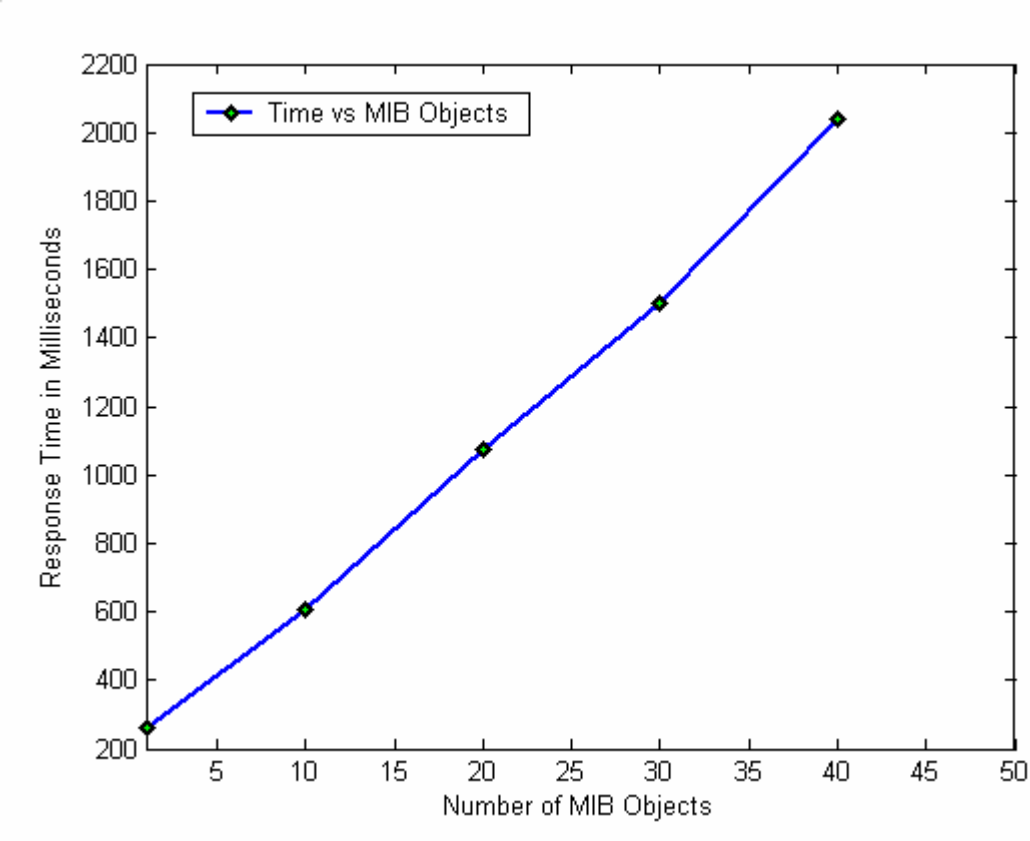

Figure 5.14: Response Time Increasing Number of MIB objects.

Table 5.6 gives the response time values for homogeneous systems, heterogeneous systems, and static allocation as the number of agent increases. Figure 5.15 shows the response time for the homogeneous vs. heterogeneous systems for the system group MIB objects in the case of equal work assignment. The experiment is conducted with two homogeneous systems and then with two heterogeneous systems. The homogeneous systems are of 350 MHz processing speed Intel Pentium II processors and the heterogeneous systems are a 350 MHz processing speed Intel Pentium II processor and a 711MHz processing speed Intel Pentium III processor. The response time for both cases

are similar because the equal work assignment does not consider the processing speed of the slave JPVM gateways. The equal work assignment will not give better performance with heterogeneous systems i.e., systems having different processing speed capacity. In case of heterogeneous systems higher capacity processors are underutilized in the case of heterogeneous systems. The quantitative results for distribution of tasks have been shown in Table 5.4 for none working master JPVM. The quantitative results for distribution of tasks have been shown in Table 5.5 for working master.

Table 5.4: Quantitative Results for Distribution of Tasks for None Working Master

Number of Distribution Tasks	Percentage of Reduction
2	71%
3	85%

Table 5.5: Quantitative Results for Distribution of Tasks for Working Master

Number of Distribution Tasks	Percentage of Reduction
2	70%
3	83%
4	88%

Table 5.6: Response Time values for Homogenous systems, Heterogeneous systems, and Static weighted load balancing

	350-JPVM	711-JPVM	HOMO	HETRO	STATIC	STATIC	
Agents	SNMP_COM	SNMP_COM	SNMP_COM	SNMP_COM	SNMP_COM	350	711
1	1016.5	609.36	739.2	608.9	612.8	0	1
10	2174.1	1131.48	1694.8	1645.5	1244.2	3	7
20	4196.2	2369.36	2451.8	2283.1	1592.6	7	13
30	6697.5	3575.48	3637.2	3236.9	2752	10	20
40	10297.8	5226.72	4813	4128.8	3739.4	13	27
50	13405.3	6780.92	5451.8	4483.6	4953.2	17	33
60	18025.8	8949.72	6784	5749.1	6079	20	40
70	20401.4	9885.44	8836.6	7285.4	7064.4	23	47
80	25054.9	13035.88	10511.4	8956.9	8334	27	53
90	36357.2	18237.72	12123.6	11071.2	9782	30	60
100	44016.3	21733.76	15065.6	13507.5	11632.6	33	67
110	52698.6	25692.52	17280.8	15703.6	12035.2	37	73
120	61505.4	29940.32	19528	17794.8	13567.6	40	80
130	69715.1	33717.2	21881.4	19020.4	15696.8	43	87
140	78662.1	39770.32	22330	20245.2	19638.2	47	93
150	89115.1	42279.28	24779.4	23554.1	22195.8	50	100
160	98816	46633.88	27225.2	25744.9	24771.8	53	107
120	61505.4	29940.32	19528	17794.8	13567.6	40	80
170	111015.5	52791.44	31415.2	28009.3	26704.6	57	113
180	126682.25	59818.32	38639.6	35369	29342.2	60	120
190	145066.6	65247.44	41670.2	38172.9	32536.8	63	127
200	150031.8	70516.16	46034.2	42122.9	35984.2	67	133

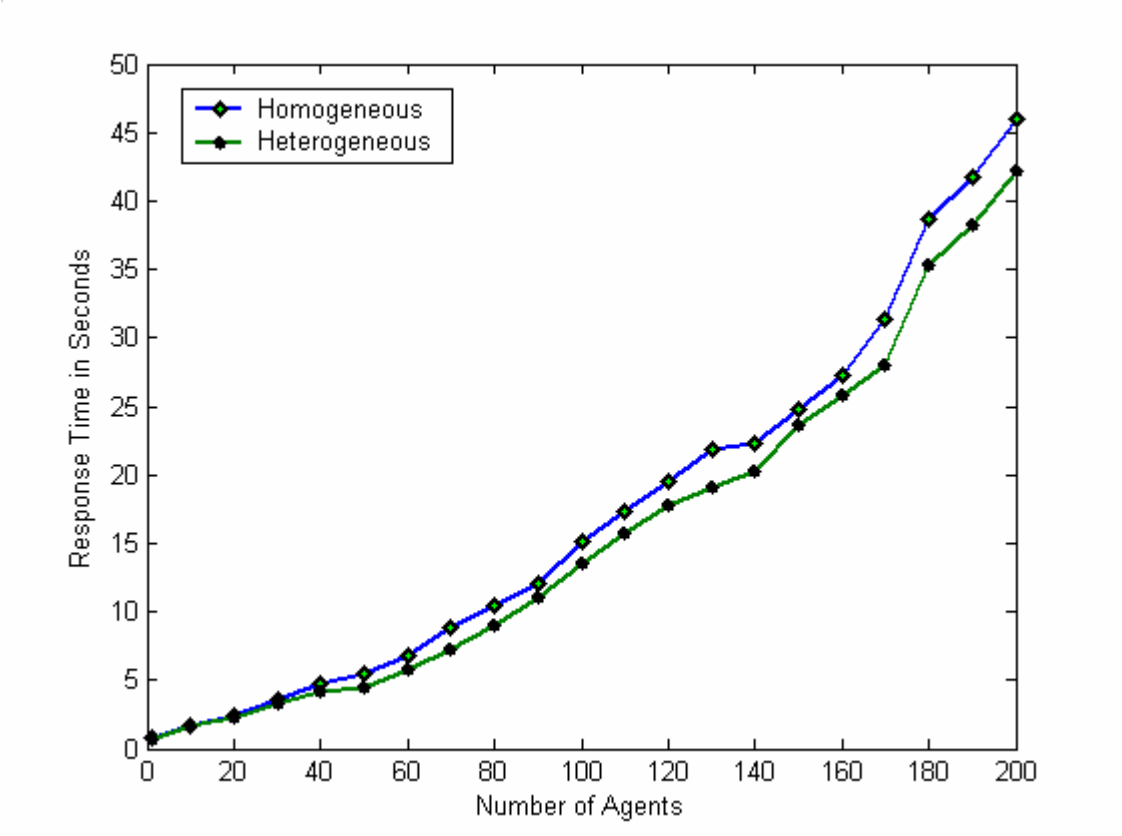

Figure 5.15: SNMP Communication Time for Homogeneous and Heterogeneous Systems.

Figure 5.16 shows the response time for heterogeneous system vs. static weighted load balancing for values in Table 5.6. In the case of the static weighted load balancing approach, if we consider an XML-based request with 30 agents that has 2752 milliseconds as the response time, then the allocation of the work to each JPVM gateway is 10 and 20 respectively for the 350 MHz and the 711 MHz. In the homogeneous systems, the 350 MHz PC takes 6697.5 milliseconds time when requesting 30 agents, the 711 MHz PC takes 3,575.58 milliseconds response time when requesting 30 agents. The heterogeneous

systems are taking 3236.9 milliseconds response time, which is equal to requesting 15 agents by the 350 MHz PC. The static weighted load balancing will take 2,752 milliseconds response time, which is equal to requesting 20 agents by the 711 MHz PC in addition to the communication time for data packing and unpacking due to the existence of two slave JPVM gateways.

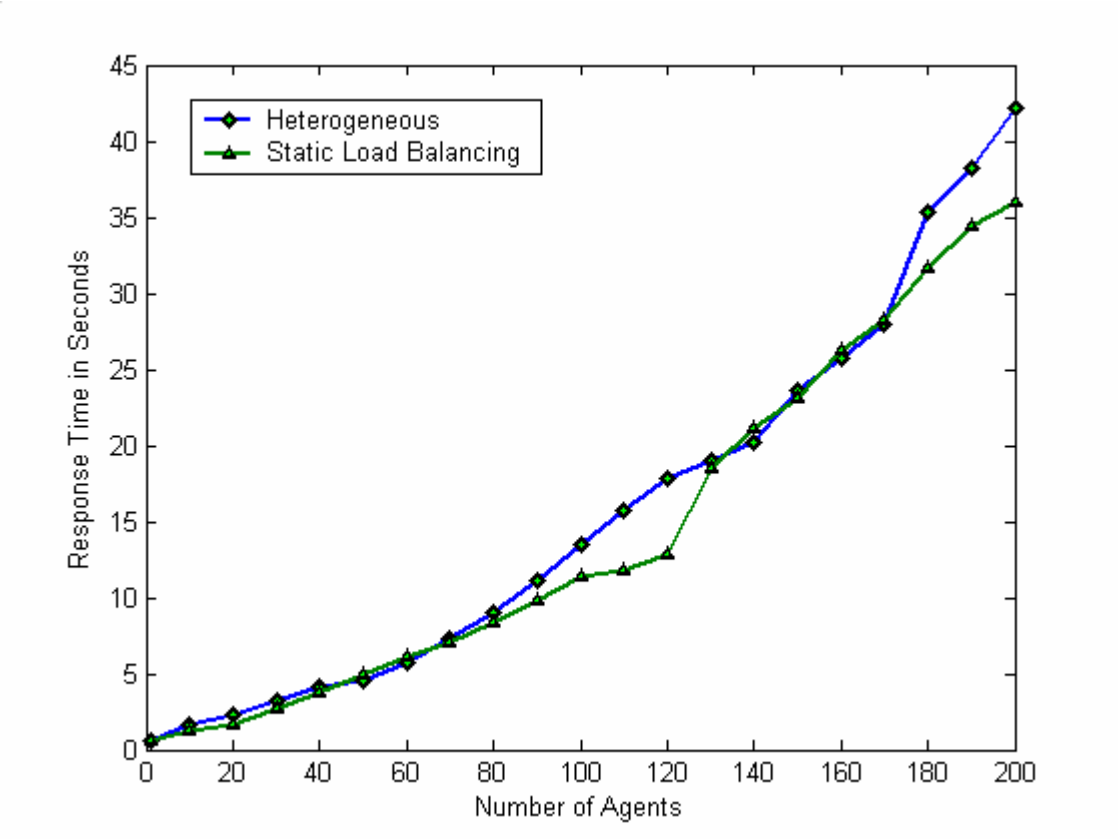

Figure 5.16: Response Time for Heterogeneous and Static Weighted load balancing

As the number of agents increases, the response time in the case of heterogeneous systems with equal work will be dominated by the lower processing speed processor. As for the case of static weighted load balancing, initially it will be good but gradually as the

number of agents increases the response time will be dominated by the higher processing speed processor. There will be a little improvement in the response time with the static weighted load balancing compared to the equal work approach.

Hence, in the case of static weighted load balancing with heterogeneous gateways, as the number of agents increases the processor with the lower processing speed gets less work and is underutilized. In the case of equal work allocation with heterogeneous gateways, as the number of agents increases the processor with the higher processing speed gets less work and is underutilized.

Figure 5.17 shows the response time for dynamic load balancing with two slave JPVM gateways. The dynamic load balancing with two slave JPVM is shown with an increasing block size of 5, 10, 20, and 50. The response time with block size 5 is higher compared to block size 10 and 20 due to the communication overhead. As the block size increases over 30 the response time increases due to the unbalanced load among the slave JPVM gateways. The unbalance occurs when the last processor executes the last work block. The response time with block size 50 as shown in Figure 5.17 has higher response time compared to block size 10 and 20. Hence, a lower block size (5-30) has a better response time compared to a higher block size.

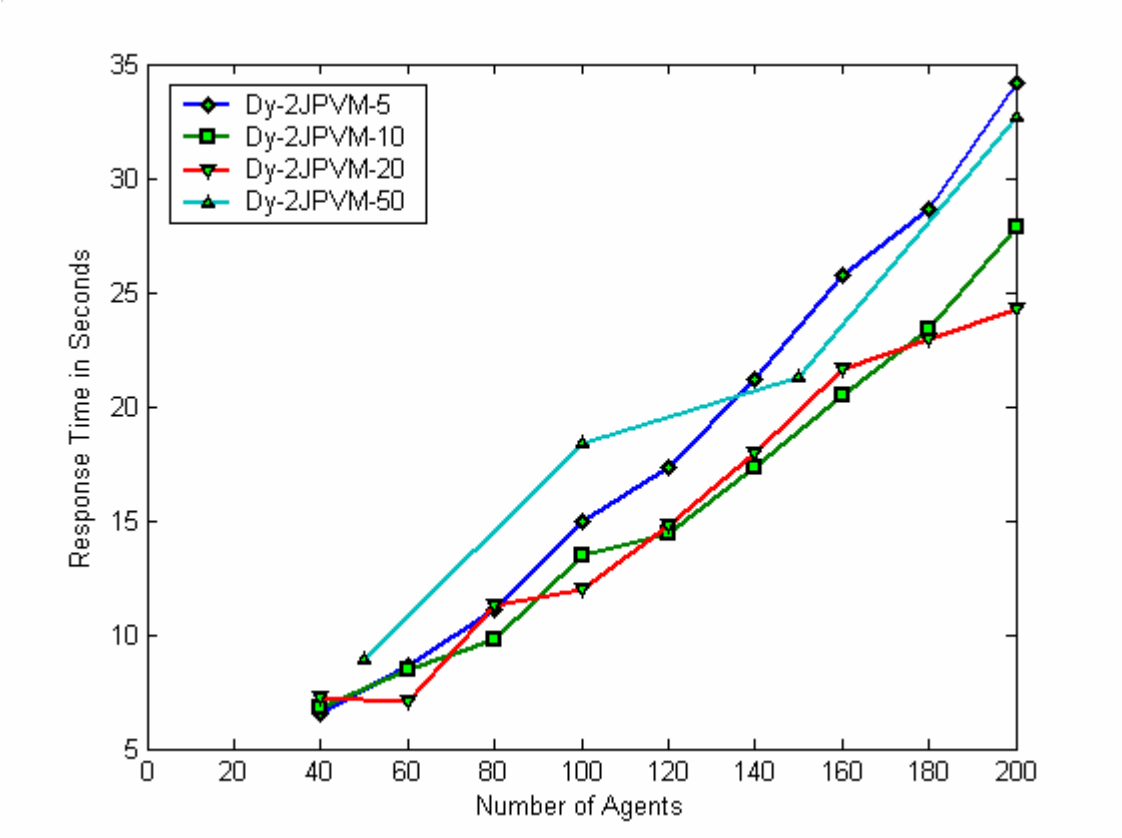

Figure 5.17: Dynamic Response Time with increasing Block Size

Figure 5.18 shows the response time for dynamic load balancing with increasing block size and static weighted load balancing with two slave JPVM gateways. The response time with block size 5, 10, and 20 is lower compared to static weighted load balancing. The response time with block size 50 is higher and approaching the response time with block size 5. The response time is better with dynamic load balancing compared to static weighted load balancing in the case of a lower block size. The increase in the block size increases the response time due to the unbalanced load among the slave JPVM gateways.

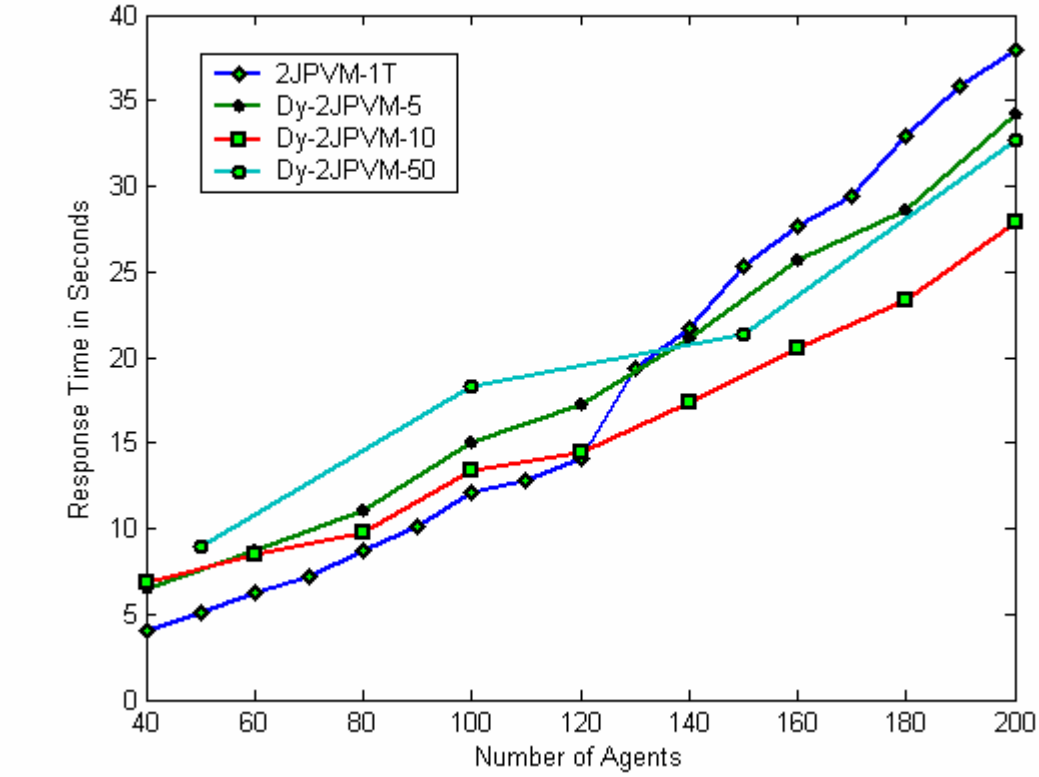

Figure 5.18: Response Time for Static and Dynamic Load Balancing

Figure 5.19 shows the dynamic response time for two and three slave JPVM gateways with block size 5, 10, and 20. The response time is shown only for the block sizes with lower response times. As the block size increases the response time increases for higher block sizes.

Hence, the response time of dynamic load balancing with a smaller block size is better compared to equal work and static weighted load balancing. The dynamic load balancing

allocates an optimal number of agents for each slave JPVM gateway to achieve the maximum efficiency.

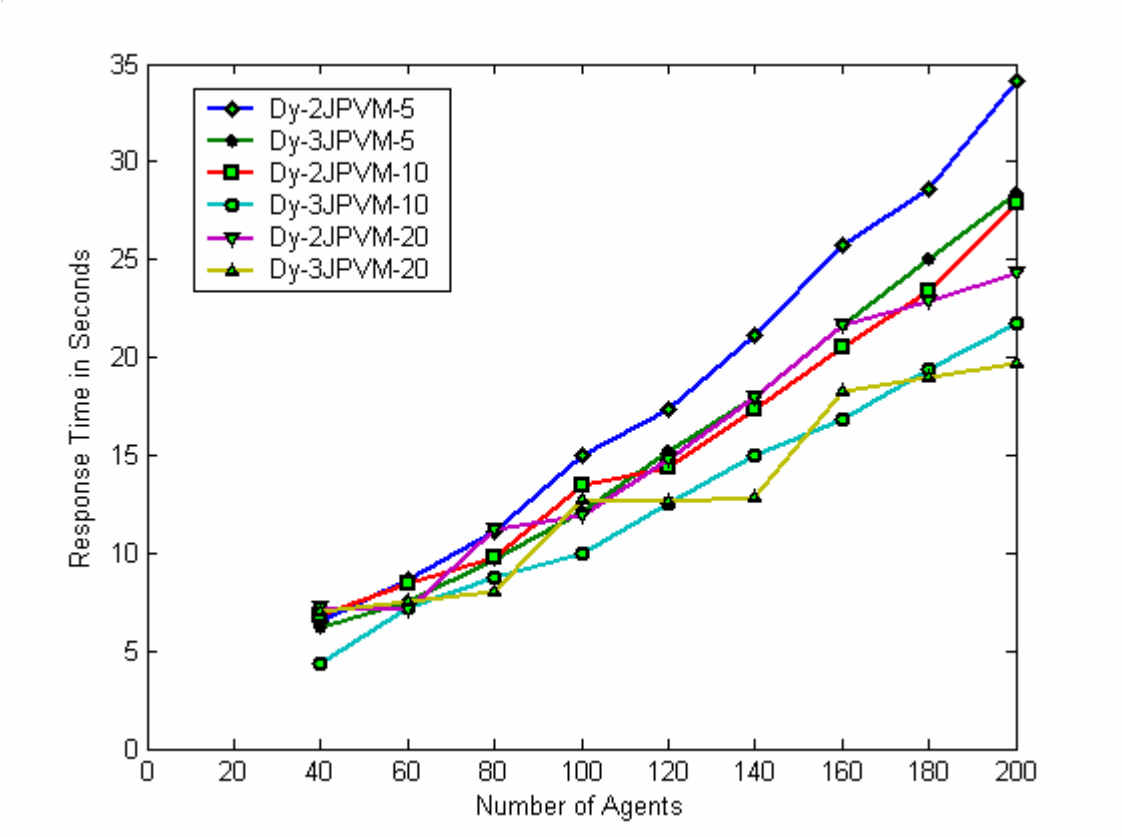

Figure 5.19: Response Time for Dynamic Load Balancing with increasing Block Size and Processors

5.3.3. Parallel Component Evaluation

Parallel algorithms divide a program into parts so that a number of processors can work on the problem at the same time. In an ideal situation, *n* processors should speed up a program so that it is completed in *(1/n)* of the time taken by a single processor. All the

problems cannot be divided into perfectly even work, and communication is required between the processors. This communication can reduce the speedup significantly. The amount of parallelism exhibited by a problem can greatly determine the speedup that a parallel implementation will offer.

The increase in computation speed from parallel implementations of problems is described using the Amdahl's Law. It is a law governing the *speedup* of using parallel processors on a problem, versus using only one serial processor.

5.3.3.1. Speedup

The speed of a program is the time it takes the program to execute. This could be measured in any increment of time. Speedup [34] is defined as the time it takes a program to execute in serial (with one processor) divided by the time it takes to execute in parallel (with many processors). Let $T(N)$ be the time required to complete the task on N processors and $T(1)$ be the time required to execute the task on single processors. The speedup $S(N)$ is the ratio as given below.

$$S(N) = \frac{T(1)}{T(N)}$$

In many cases the time $T(1)$ has, as noted above, both a serial portion T_s and a parallelizable portion T_p. The serial time does not diminish when the parallel part is split up. If one is "optimally" fortunate, the parallel time is decreased by a factor of $(1/N)$. The speedup becomes as below.

$$S(N) = \frac{T(1)}{T(N)} = \frac{T_s + T_p}{T_s + T_p / N}$$

The above elegant expression is known as *Amdahl's Law* and is usually expressed as an inequality. This is in almost all cases the *best* speedup one can achieve by doing work in parallel, so the real speed up $S(N)$ is less than or equal to this quantity.

Table 5.7 shows the speedup achieved with increasing number of parallel tasks running on the single slave JPVM gateway with equal work to all the slave JPVM. The speedup with increasing number of tasks has a linear increment in speedup.

Table 5.7: Speedup with increasing number of Tasks

Number of Tasks	Speedup	Efficiency
2	1.71766	0.85883
3	2.418656	0.806219
4	2.843945	0.710986
5	3.061021	0.612204

Table 5.8 shows the speedup achieved with increasing number of slave JPVM processor or slave Table 5.8 gateways, with equal work to all the slave processors. The graph for the increasing number of slave JPVM gateways and increasing number of JPVM tasks running on a single JPVM gateway is shown in Figure 5.20 . We have observed linear speedup in the case of increasing slave JPVM tasks on a single gateway and super linear

speedup in the case of increasing the number of JPVM processors with one task on each processor.

Table 5.8: Speedup with increasing number of Processors

Number of Processors	Speedup	Efficiency
2	3.395015	1.697508
3	6.146013	2.048671
4	8.670053	2.167513

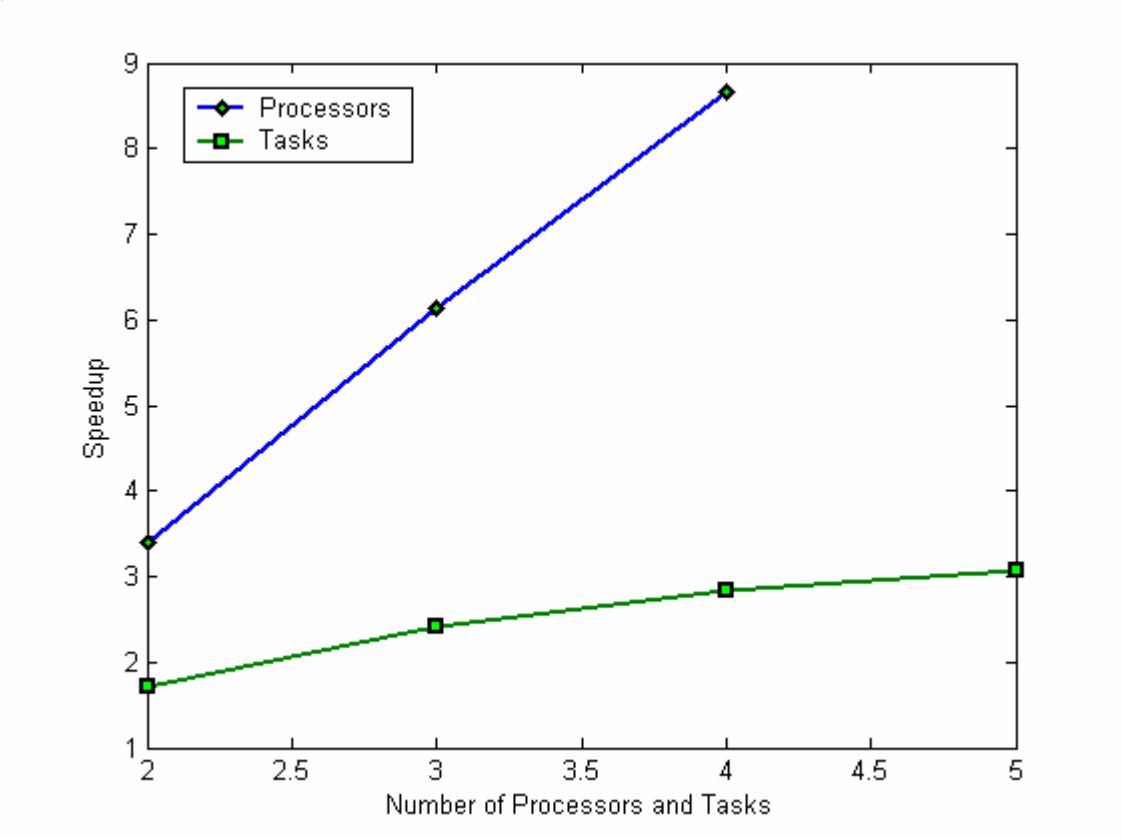

Figure 5.20: Speedup with increasing number of Processors and Tasks

5.3.3.2. Efficiency

The efficiency [34] of a parallel program is defined as the speedup, divided by the number of processors used in the parallel execution.

$$E = S(n)/n$$

Where, $S(n)$ is speedup with n parallel tasks. Column four in Table 5.7 and Table 5.8 shows the efficiency values respectively increasing the number of tasks running on single slave JPVM gateway and increasing the number of slave JPVM processors. Figure 5.21 shows the efficiency with varying number of JPVM tasks running on a single slave gateway and varying number of slave JPVM processors for the efficiency values given in Table 5.7 and Table 5.8. The efficiency decreases by either increasing number of tasks running on a single slave JPVM gateway or increasing the number of JPVM processors.

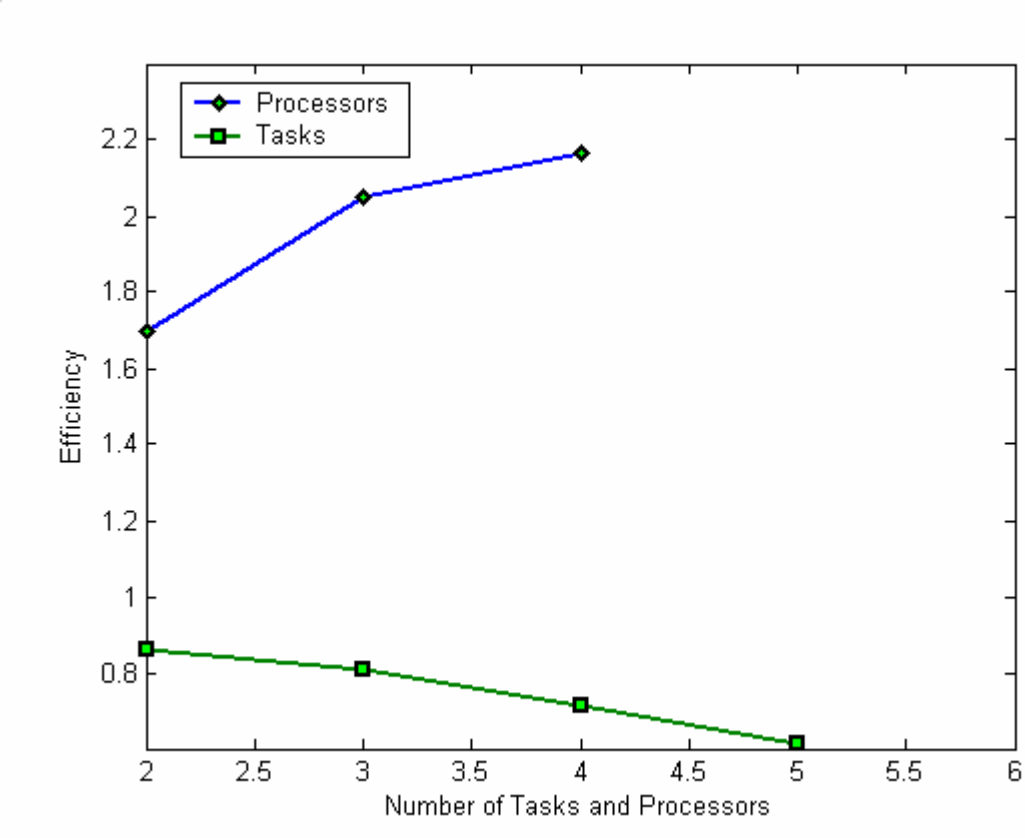

Figure 5.21: Efficiency with increasing the Number of Tasks and Processors.

5.3.4. Network Traffic

Figure 5.22 shows the network traffic between the XBM manager and SNMP agents through the gateway. The traffic can be divided into two components, traffic between the XML-based manager and the gateway, and traffic between the agents and the gateway. The traffic between the gateway and the agents is only due to the SNMP communication and the traffic between the XML-based manager and gateway is due to the exchange of XML-based requests over the HTTP protocol. The graph shows a linear increment in the traffic between the XBM manager and the gateway, and the traffic between the SNMP

agents and gateway. All the three approaches generate the same amount of network traffic.

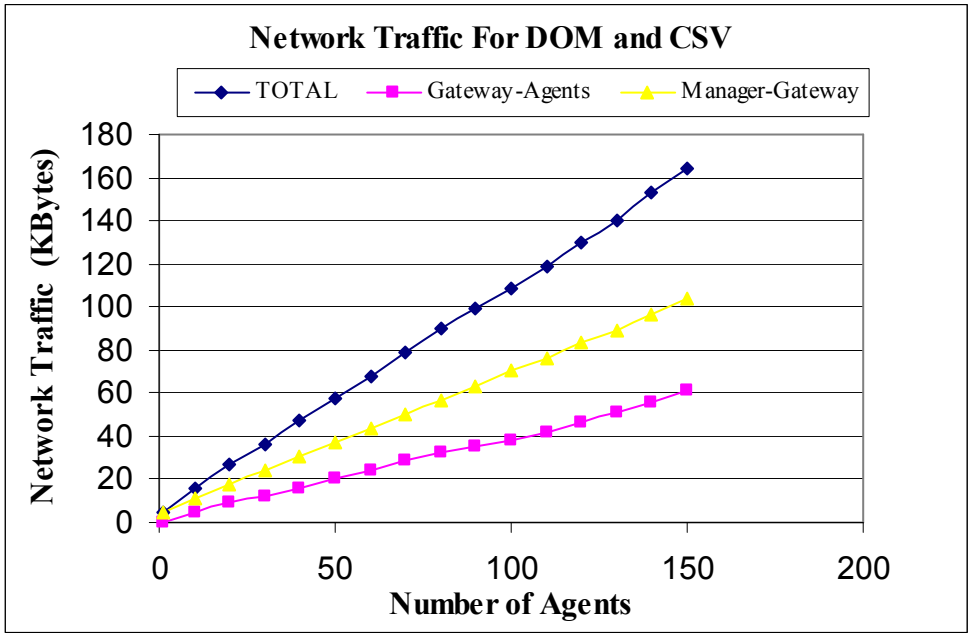

Figure 5.22: Network traffic of DOM, CSV and JPVM-NM of System Group

In the case of JPVM, the traffic between the manager and the slave JPVM gateways remains the same. The traffic between the slave gateways and the SNMP agents will be distributed based on the number of slaves.

5.3.5. Message Size

The message length (size) refers to the length of the XML-based SNMP message. A single SNMP request contains only one agent and may contain more than one OID. A multirequest contains multiple agents and more than one OID for each agent. A multi

request can be thought of as a collection (bundle) of single requests as shown in Figure 5.23.

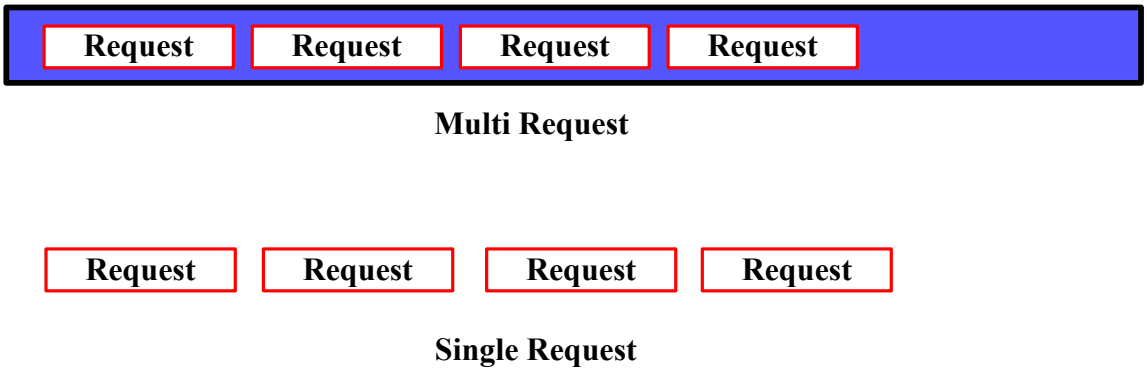

Figure 5.23: Multi Request and Single Request Format.

The message size can be classified into two types based on the request message size and the response message size.

1. Multiget request message size: It is the size of a multiget request message PDU. It includes multiple agents followed by a list of OIDs and other communication parameters like: operation type, community, version, etc.
2. Multiget response message size: It is the size of a multiget response message PDU for the given get request PDU. This PDU contains the response message received from the agents for the corresponding MIB objects.

Table 5.9 shows the size for a multiget request message. It presents the message length for the XML-based request of the sysDescr and sysContact MIB objects. It provides a comparison of the message for Multihostget and Multiobjectget, for the legacy SNMP

based method, the POSTECH's XBM and the POSTECH's XML/SNMP gateway based request.

Table 5.9: Message Size of Multiget Request

Management Property	Get Request Message Size in Bytes				
	SNMP	XBM	XML/SNMP POSTECH	XML/SNMP, Single DOM, CSV and JPVM	
				MGH	MGO
SysDescr	82	508	666(584+82)	356(274+82)	341(259+82)
SysContact	83	510	678(586+82)	359(276+83)	344(261+83)

Table 5.10 shows the size for a Multiget request message with one and ten agents. It presents the message length for the XML-based request for Multihostget and Multiobjectget with 1 and 10 agents in the request. The Multihostget has much less request size compared to the Multiobjectget as the number of agent increases in the XML-based multi request. The advantage with the Multihostget is that the request size increment decreases as the number of agent increases. The advantage of the Multihostget is that it allows to use the same set of MIB objects for multiple agents. The advantage of the Multiobjectget is that it can support a variable number of MIB objects in the request for each agent.

Table 5.10: Message Size of Multiget Request with one and ten agents

Management Property	Get Request Message Size in Bytes			
	XML/SNMP, Single DOM, CSV and JPVM			
	MGH		MGO	
Agents	1	10	1	10
SysDescr	356(274+82)	1420(600+820)	341(259+82)	3410(2590+820)
SysContact	359(276+83)	1428(598+830)	343(261+83)	3440(2610+830)

CHAPTER 6

CONCLUSION AND FUTURE WORK

Conclusion

SNMP has been widely used for monitoring network devices for the last 15 years due to its simplicity. But, it is not successful in few areas of network management, and one such area is configuration management. Since, SNMP-based network management does not meet the current network management requirements, there have been many evolutionary approaches to improve on the SNMP framework. One of such evolutionary approach is the use of XML. However, Network management based on XML has also few drawbacks, particularly the processing overhead of the XML-based requests. Our work's objective is mainly to improve the processing speed of the XML-based network management operations.

In this thesis, we extended the work of POSTECH in the area of XML-based network management. The framework we described allows a manager to access multiple agents. We defined new types of messages that could be sent by a manager, namely Multi-Get-Request, Multi-Set-Request, and Response. These messages can be widely used in configuration management. The implementation for Multi-Get-Request and Multi-Set-Request can be achieved through an HTTP-based interaction method and a SOAP-based interaction method. We described how a manager can send in one message either one request to multiple agents, multiple requests to one agent, or multiple requests to multiple

agents. The proposed single DOM Tree-based approach, CSV-based approach, and JPVM-based approach are evaluated and the performances of these frameworks are compared with the recent work on the XML-based network management.

The single DOM tree–based approach has been used in the literature with XML for network management, but it is time consuming. CSV is a very simple and well known format that has not been used with XML for network management. We presented a novel approach that makes use of CSV in XNM. The comparison of these two approaches shows that the CSV approach outperforms the DOM approach and provides ~50% response time savings.

The JPVM-based approach has been used to achieve the distribution of management tasks. In this approach, we divide the management work into a number of tasks that can be assigned to a number of slave JPVM gateways. We can also have a number of tasks running on the same slave JPVM gateway. With JPVM, we achieved distribution and parallelism.

The experimental results show that the JPVM-based approach running with a number of slave JPVM gateways gives better results compared to the approach where a number of tasks are running on a single slave JPVM gateway.

The JPVM-based approach has been implemented with equal work, static weighted load balancing and dynamic weighted load balancing. The equal work approach gives better results with homogenous slave JPVM gateways, whereas the static weighted load balancing gives a sub optimal response time with heterogeneous slave JPVM gateways.

The dynamic weighted load balancing gives better results with smaller block sizes compared to the equal work and static weighted load balancing approaches.

Future Work

The results obtained in this thesis show that XML-based network management is an area that can enhance the existing network management paradigms. We have contributed to this area with many new approaches, but we believe that more work can be done to improve even more on this work. In this section, we list some of our recommended future research directions:

- The single DOM Tree-based approach can be improved by having multiple lightweight DOM tree document fragments of the XML-based request.
- In the case of static weighted load balancing algorithm we have taken only the processing speed of the slave JPVM as the metric to assign work to the slave JPVM gateways. The performance of the algorithm can be still improved by considering the other various parameters such as the current load on the processor, the number of current threads running on the processor and the current network bandwidth available for transmission over the network. We will get realistic results by considering the above parameters for the assignment of the weights to the slave JPVM gateways.
- Besides the Internet management community, there are many technologies developed and are excellent for the Internet management, one of such interesting technology is web services. It provides a single uniform software infrastructure to

support a wide range of distributed services. The World Wide Web Consortium (W3C) has standardized web services. Research in this area has just begun and one can investigate its merits in network management. Hence, these frameworks can be extended to web services through SOAP, Web Services Description Language (WSDL) and Universal Description Discovery and Integration (UDDI) technology.

- XForms (XML Forms) are an upcoming XML technology. They are the next generation web forms, and can be used at the XML-based manager to improve the efficiency of processing the XML-based request at the gateway. These XForms can also be used in conjunction with SOAP, WSDL and UDDI for web services based network management.
- The JPVM-based approach can be extended with multiple master JPVM gateways, where a manager communicates with multiple master JPVM gateways. And, each master JPVM gateway communicates with a number of slave JPVM gateways. This way, we can have a hierarchy of XML/SNMP JPVM gateways.
- The JPVM-based approach can be extended with multiple slave JPVM gateways, running multiple JPVM tasks. The manager can then send work to multiple slave JPVM gateways, each of which is running multiple JPVM tasks.
- Predictive dynamic load balancing: Study the behavior of the slave JPVM gateways and then assign the load by predicting the capacity on the slave JPVM gateways.

- Adaptive dynamic load balancing: first assign the load based on the processing speeds of the JPVM gateways and then adapt it based on the response times of the slave JPVM gateways.If we have agents with different SNMP versions and different community names then the multirequest can be extended to support multiple multirequests based on the version and community names.

REFERENCES

[1] J.P.Martin-Flatin, “Push vs. pull in web-based network management”. In Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99), pages 3-18, USA.

[2] J.P.Martin-Flatin, Laurent Bovet and Jean-Pierre Hubaux, ”JAMAP: a Web-Based Management Platform for IP Networks”, Proceeding 10th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Zurich, Switzerland, October,1999.

[3] The Simple Times, “The Quarterly Newsletter of SNMP Technology, Comment, and Events (sm)”, Volume 4, Number 3 July, 1996 http://www.simple-times.org/pub/simple-times/issues/4-3.html

[4] James Won-Ki Hong; Ji-Young Kong; Tae-Hyoung Yun; Jong-Seo Kim; Jong-Tae Park; Jong-Wook Baek; ”Web-based Intranet services and network management“, *Communications Magazine, IEEE,* Volume: 35 Issue: 10, Oct. 1997, Page(s): 100–110.

[5] Jeong-Hyuk Yoon, Hong-Taek Ju and James W. Hong, "Development of SNMP-XML Translator and Gateway for XML-based Integrated Network Management", *International Journal of Network Management (IJNM)*, Vol. 13, No. 4, July-August 2003, pp. 259-276.

[6] Mi-Jung Choi, Hyoun-Mi Choi, Hong-Taek Ju and James W. Hong, "XML-based Configuration Management for IP Network Devices", Special Issue on XML-based Management of Networks and Services in *IEEE Communications Magazine*, Vol. 41, No. 7, July 2004. pp. 84-91. (SCI).

[7] J.P.Martin-Flatin, " Web-Based Management of IP Networks and Systems", Wiley series in communications Networking and Distributed Systems, 2003.

[8] William Stallings, "SNMP, SNMPV2, SNMPv3, and RMON 1 and 2", 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999.

[9] Mani Subramanian, "Network Management: Principles and Practice", Addison-Wesley, Hardcover, Published December 1999, 644 pages, ISBN 0201357429.

[10]J.P Martin- Flatin, Aiko Pras, Jurgen Schonwalder, "On the Future of Internet Management Technologies", *IEEE Commmunication Magzine,* Vol. 41, No. 10, *October 2003*.

[11]W3C, "Extensible Markup Language (XML) 1.0", *W3C Recommendation*, October 2000.

[12]Brett McLaughlin, "Java and XML", *O'Reilly, Second Edition*, 2001.

[13]"Professional Java XML", *Wrox Publications*, 2001.

[14]W3C, " XML Schema Part0: Primer", *W3C Recommendation,* May 2001.

[15]W3C, " XML Schema Part1: Structures", *W3C Recommendation,* May 2001.

[16]W3C, " XML Schema Part2: Data Types", *W3C Recommendation,* May 2001.

[17]W3C, " XML Path Language (XPath) Version2.0", *W3C Working Draft*, April 2002.

[18]W3C, " XQuery and XPath Functions and Operators Version2.0", *W3C Working Draft*, April 2002.

[19]W3C, " XML Query Language (XQuery) ", *W3C Working Draft*, April 2002.

[20]W3C, "Document Object Model (DOM) Level 2 Core Specification ", *W3C Recommendation*, November 2000.

[21]W3C, “Document Object Model (DOM) Level 2 Traversal and Range Specification”, *W3C Recommendation*, November 2000.

[22]Hyoun-Mi Choi,Mi-Jung Choi, James W.Hong, “XML-based Network Management for IP Networks”, *ETRI Journal,* Volume 25, Number 6, December 2003.

[23]Frant Strays, Torsten Klie, “Towards XML Oriented Internet Management”, www.ibr.cs.tu-bs.de/papers/im-2003.pdf

[24]Straus, F. “A library to access SMI MIB information”, http://www.ibr.cs.tubs.de/projects/libsmi/

[25]Phil Shafer “XML-Based Network Management” – White Paper, *Juniper Networks, Inc.*, 2001, http://www.Juniper.net/solutions/literature/white_papers/200017.pdf

[26]Sqalli H.M., Sirajuddin S., “Extensions to XML based Network Management”, (ICICS-2004) *2nd International conference on information and computer sciences,* Dhahran, KFUPM, November 2004.

[27]Apache Software Foundation, “The Jakarta Project - Tomcat,” http://jakarta.apache.org/tomcat/.

[28]Apache Software Foundation, “The Xerces Java XML parser”, http://xml.apache.org/xerces2-j/

[29] IReasoning Networks, “SNMP API”, http://www.ireasoning.com/products.shtml

[30]SoftPerfect Research, “SoftPerfect Network Protocol Analyzer”, http://www.softperfect.com/products/networksniffer/

[31]Adam J.Ferrari, “JPVM: The Java Parallel Virtual Machine”, *Technical Report* CS-97-29 VA 22903, USA, December 8, 1997 http://www.cs.virginia.edu/jpvm/

[32]Rajesh Subramanyan, Jose Miguel-Alonso, and Jose A.B Fortes,“ A Scalable SNMP-based Distributed Monitoring System for Heterogeneous Network Computing”, *proceedings of the 2000 ACM/IEEE conference on Supercomputing* (CDROM), 2000, Dallas, Texas.

[33]Thomas M. Chen, Stephen S. Liu, “A Model and Evaluation of Distributed Network Management”, *IEEE Journal on Selected Areas in Communications*, Volume. 20, no. 4, may 2002.

[34] Juliet A. Holwill and Sarana Nutanong,”Parallel Implementation of the Mandelbrot Set“, www.cs.mu.oz.au/~aharwood/nppc-projects/03/e/index.html

ACRONYMS

SNMP	Simple Network Management Protocol
API	Application Programming Interface
CSV	Comma Separated Values
DOM	Document Object Model
DTD	Document Type Definitions
HTML	Hypertext Markup Language
RFC	Request for Comments
SAX	Simple API for XML
UI	User Interface
XML	Extensible Markup Language
XSL	Extensible Style Sheet Language
XSLT	Extensible Style Sheet Language Transformations
XPATH	XML Path Language
XUPDATE	XML Update Language
XQUERY	XML Query Language.
JPVM	Java Parallel Virtual Machine
PVM	Parallel Virtual Machine
OID	Object Identifier
MIB	Management Information Base

VITA

Name: Shaik Sirajuddin

Date of Birth: 3rd April 1979

Nationality: Indian

Bachelor Degree: Bachelor of Technology in Computer Science and Engineering from Jawahar Lal Nehru Technological University (2001), Hyderabad, AP, INDIA.

Masters Degree: Master of Science in Computer Engineering (December 2004), from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

Papers Published

1. Sirajuddin S, Sqalli H.M, "Distributed XML-based Network Management using JPVM ", (IJNM) *International Journal of Network Management*, (Submitted).

2. Sirajuddin S, Sqalli H.M, ”Comparison of CSV and DOM Tree Approaches in XML-based Network Management”, (ICT 2005) - 12th International Conference on Telecommunications”, South Africa, June, 2005
3. Sqalli H.M., Sirajuddin S., “Extensions to XML based Network Management”, (ICICS-2004), *2nd International conference on information and computer sciences,* Dhahran, KFUPM, November 2004.