A Framework for a Reliable & Fault-Tolerant Network Management Architecture

BY

LOUAI ADNAN AL-AWAMI

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

June 2006

KING FAHAD UNIVERSITY OF PETROLUEM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by LOUAI ADNAN AL-AWAMI 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 (Chairman)

Dr. Mostafa Abd-El-Barr (Member)

Dr. Marwan Abu-Amara (Member)

Dr. Adnan Gutub
(Department Chairman)

A. Sanjum
Dr. Mohammed Al-Ohali
(Dean of Graduate Studies)

10/06/2006
Date

Dedicated to

my parents

&

my beloved wife Alaa.

Acknowledgements

All praise be to Allah Subhanahu-wa-ta-a'la for his limitless blessings. May Allah bestow peace on his prophet Muhammad (peace be upon him and his household). I acknowledge the support and facilities provided by King Fahd University of Petroleum & Minerals (KFUPM).

I would also like to express my deepest gratitude to my thesis advisor Dr. Mohammed Sqalli for his patience, guidance and sincere cooperation through-out this thesis. I also value the help and support given by my committee members Prof. Mostafa Abd-El-Barr and Dr. Marwan Abu-Amara.

My acknowledgements extend to the chairmen of the Computer Engineering Department, Dr. Sadiq Sait, Dr. Abd-Aziz Al-Mulhem and Dr. Adnan Gutub for their cooperation and providing the department facilities.

I also acknowledge the cooperation of Dr. Kishor S. Trivedi in providing us with a copy of the SHARPE tool.

I would like to acknowledge the constant support, understanding and love that I received from my wife Alaa.

I am also thankful to all colleagues with whom I shared wonderful moments and made this an enjoyable experience.

Contents

List of Tables

List of Figures

THESIS ABSTRACT

Name: LOUAI ADNAN AL-AWAMI

Title: A Framework for Reliable and Fault-Tolerant Network Management Architecture

Degree: MASTER OF SCIENCE

Major Field: COMPUTER ENGINEERING

Date of Degree: June 2006

In this work, our objective is to design and implement a three-tier Fault-Tolerant Network Management System (FTNMS). High-Availability (HA) technology is used to develop this system, where two Manager-of-Managers (MoM) are implemented at the highest level in an active-passive mode. In the middle level however, Mid-Level Managers (MLMs) are used to manage different areas of the network and relieve the MoM from dealing with individual nodes and hence enhancing the scalability of the whole Network Management Systems (NMSs). MLMs are configured to work in pairs where each pair contains two MLMs working in an active-active mode. The MoMs and MLMs have the capability of backing up each other in the case of a failure.

The proposed framework provides reliability, availability, centralized control and scalability at an affordable cost. The design uses off-the-shelf components to make it simple and robust. Without any changes to the existing management protocols and management applications, the framework can be integrated with existing network management systems to improve their reliability. Besides, the system allows easy extension of both centralized and hierarchical systems to fault-tolerant NMSs.

The system is evaluated in terms of failover time, increase in traffic, reliability and availability. The results show that the overall reliability of a centralized NMS with reliability of R_{NMS} can be improved by a factor of $1 - R_{NMS}$. Moreover, the system is shown to be able to manage events even during failover time. The thesis includes some test cases to demonstrate these results.

MASTER OF SCIENCE DEGREE

King Fahd University of Petroleum and Minerals, Dhahran.
June 2006

ملخص الرسالة

الاسم: لؤي عدنان محمد العوامي
عنوان الدراسة: إطار عمل لنظام إدارة شبكات إعتمادي و متحمل للأعطال.
التخصص: هندسة الحاسب الآلي
سنة التخرج: جمادى الأولى ١٤٢٦هـ

دف هذه الرسالة إلى تصميم و بناء نموذج لنظام إدارة شبكات ذو إعتمادية عالية و قادر على تحمل الأعطال بواسطة تكنولوجيا الإتاحية العالية, حيث يتكون النظام من جزئيين: المدير رئيسي و المدراء البينيون. يتكون الجزء الأول من مديرين رئيسين يعملان بنسق نشط-خامل حيث يكون أحد المدراء الرئيسين نشطاً بينما يكون الآخر خاملاً, و يكمن للمدير الخامل أن ينشط عند حدوث خلل ما في المدير النشط. أما الجزء الثاني فيتكون من عدة مدراء بينيين مقسمين كأزواج و يتكون كل زوج من مديرين بينيين يعملان على نسق نشط-نشط و يمكن لكل منهما تولي المهام الموكلة لقرينه حال حدوث عطل ما الآخر. كما يعمل كل مدير بيني على إدارة جزء مختلف من الشبكة و تخفيف الأعباء الموكلة للمدير الرئيسي.

يتيح النموذج المقترح إعتمادية و إتاحية عاليتين مع قابلية للتدرج و المحافظة على مركزية التحكم و بتكلفة ضئيلة. كما يستخدم النموذج مكونات معروفة و متوفرة مما يجعل النموذج سهل و قوياً. بالإضافة إلى ذلك, يمكن دمج النظام المقترح مع الأنظمة الموجودة لتحسين إعتماديها دون الحاجة إلى تغير البرمجيات أو البروتكولات المعمول ا. و يتيح النظام تطوير الأنظمة المركزية إلى أنظمة هرمية.

و قد تم تقييم النظام من حيث الوقت المطلوب للإستعادة التشغيل حال حدوث أعطال, و الزيادة في الحزم المنقولة في الشبكة التي يتسبب ا النظام بالإضافة إلى قياس الإعتمادية و الإتاحية. و توضح النتائج مدى الإعتمادية العالية التي يوفرها النظام حتى لو كان مكوناً من أجزاء ذات إعتمادية منخفضة. و تبين النتائج أن النظام قادر على مواصلة دوره حتى خلال عملية إنتقال المسؤولية من المدير ذو الخلل إلى قرينه السليم. و تتضمن الرسالة عدة سيناريوهات لتوضيح إيجابيات النموذج المقترح.

Chapter 1

Introduction

As computer networks grow in size, complexity, heterogeneity, services and users, managing them to guarantee a high level of operation, availability, security and performance become more challenging [2]. Network management can be defined as “administrative services for managing a network, including configuring and tuning, maintaining network operation, monitoring network performance, and diagnosing network problems” [3].

A Network Management System (NMS) is important in everyday adminstration of computer networks and helps in easing the management by automating some tasks such as configuration and faults reporting. An (NMS) is meant to manage a network even during critical failure situations. This requires the NMS itself to be reliable and fault-tolerant. Centralized network management has been shown to be inadequate for today’s networks’ needs since it lacks dependability, scalability and performance. This leads - in turn - to seeking better approaches such as hierarchical and distributed network management.

The reliability of NMSs has not been addressed well in the literature and only little work can be found [4][5][6]. The main drawbacks of the existing work can be summarized as follows:

1. It does not address neither how databases are duplicated between managers nor how

managers can have access to each others' databases when taking over.

2. The introduction of the new systems requires some changes either at the protocol level or at the application level, i.e. manager or agent.

3. The solutions do not offer a centralized way for viewing and controlling the system or the network.

4. Some solutions are expensive in terms of hardware such as NetKeeper [5].

5. The solutions do not show the trade-offs such as the increase in traffic due to exchanging information between nodes, time needed for take over to complete, etc.

As technologies are advancing, it becomes possible for new paradigms to be implemented. Many technologies have contributed to the recent development in hierarchical and distributed network management such as Web-Based Network Management, Distributed Object Computing (DOC), Policy-Based Network Management, Java-Based Network Management, XML-Based Network Management, Code Mobility for Network Management, Intelligent Agents, Active Networks and Economic Theory. However, other technologies are still to be exploited.

In this thesis, a new framework for solving the reliability of network management systems is proposed. The system provides a way to extend the centralized and the hierarchical models into reliable ones without changing the protocols or the applications used. The design uses Mid-Level Managers (MLM) to assist the Manager-of-Managers (MoM) and to provide better scalability. The MoM is fault-tolerant and consists of two managers in an active-passive mode.

The work also proposes high-availability (HA) technology as a new enabling technology for future NMSs. The two nodes in the MoM use the heartbeats mechanism to monitor each other. Whenever the active node fails, the backup node switches to active and continues man-

aging the network. A similar innovative pairing-based redundancy mechanism is proposed between each pair of MLMs but with an active-active operation.

The framework can be integrated transparently into existing centralized and hierarchical NMSs to make them fault-tolerant. It also provides all the above features at zero cost software and an affordable hardware cost. Besides, the design allows for a centralized view and control of the whole network without loosing database redundancy.

The MLMs were written in Java. A virtual testbed using User-Mode Linux (UML) was used. Linux Heartbeat package was used for heartbeats exchange. The databases between nodes were mirrored using Distributed Redundant Block Device (DRBD). The system was tested for some functionalities such as the Remote Operation (Remop) which is part of the Distributed Management (Disman) work.

In the design, many issues are addressed and solutions are proposed including IP addressing and takeover procedure. Different scenarios were analyzed to demonstrate the features of the design. A performance, reliability and availability analysis was also conducted using mathematical models and results are discussed.

The results show a large increase in reliability and availability compared to ordinary centralized and hierarchical systems. The only cost comes from the increase in traffic due to the synchronization process between different nodes. Yet, it is possible to reduce the traffic by proper placement of nodes in the network.

The thesis is organized as follows: Chapter 2 is devoted to an overview of network management, Chapter 3 presents some background material on fault-tolerance and high-availability. Chapter 4 discusses the related work in the area of network management reliability. The proposed framework design, implementation and testing are discussed in Chapter 5, 6 and 7, respectively. Chapters 8 discusses a mathematical analysis for performance, reliability and availability of the proposed system.

Chapter 2

Network Management

The work presented in this thesis is of a multidisciplinary nature and requires some background in various areas. Hence, this chapter and the following chapter present some overview of the background material from different areas we deal with. In this chapter, some network management fundamentals are introduced. The discussion covers different network management models, operations and paradigms. The discussion includes also an overview of Remote Operation (Remop) which is part of the Distributed Management (Disman).

2.1 Overview of Network Management

The network management model defined by the International Standard Organization (ISO) describes network management as being composed of four models: an organization model, an information model, a communication model and a functional model. A pictorial representation of those models can be seen in Figure 2.1 [1].

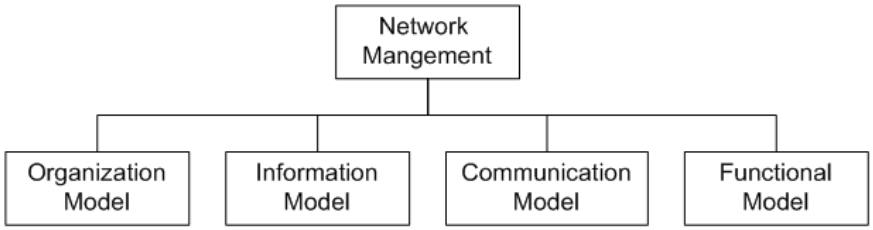


Figure 2.1: ISO Network Management Model

1. **Organization Model**

   The organization model describes the components of a network management system and their relationships. Figure 2.2 shows a simple two-tier organization model. It defines the term *object* to refer to the network element being managed such as a switch, a router, a workstation or a server process. An *agent* is a management process running on an object. Objects that can run an agent process are called *managed objects*. On the other hand, objects that can not run such a process are called *unmanaged objects*. The *manager* manages objects by retrieving information from the agent residing on them, processing and storing them in the database. The agent can also send unsolicited messages called *traps* or *notifications* when a specific *alarm* is matched.

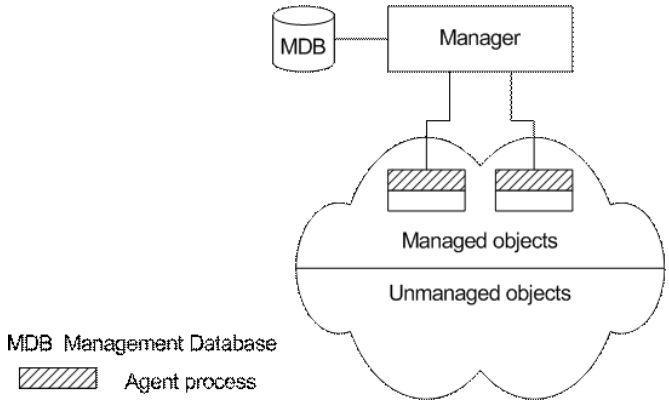


Figure 2.2: Two-Tier Network Management Organization Model

2. **Information Model**

   The Information Model specifies how the management information is structured and represented. Such specifications allow different management entities to exchange management information. It includes the *Structure of Management Information (SMI)* which defines the syntax and semantics of management information. It also includes the *Management Information Base (MIB)* which deals with the relationships and storage of management information.

3. **Communication Model**

   The Communication Model deals with how information is exchanged, e.g. the transport medium, the messages types and format. Figure 2.3 shows the communication model. The manager sends *requests* to the agent and receives *responses*. The agent can also send traps or notifications to the manager. Managers can also exchange messages known as *inform requests* in SNMPv2.

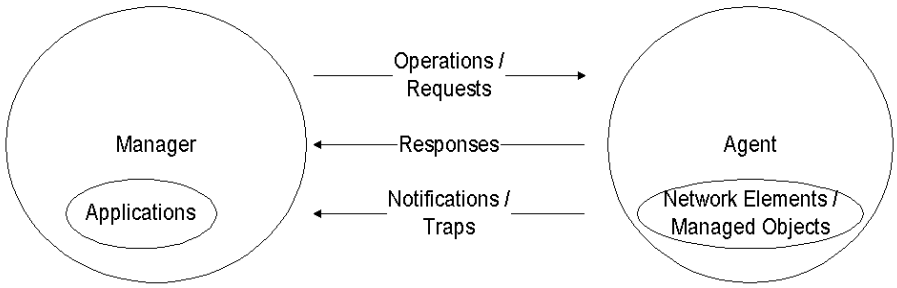


Figure 2.3: Communication Model.

4. **Functional Model**

   The Functional Model deals with the user-oriented aspects of network management. According to the ISO, the following functional areas are included in the functional model:

   (a) Configuration Management

(b) Performance Management

(c) Fault Management

(d) Accounting Management

(e) Security Management

Configuration Management deals with setting and changing the configurations of network components. **Performance Management** addresses the performance behavior of the network by collecting statistics about traffic, delay, etc. **Fault Management** is concerned with detection and isolation of faults in the network. **Accounting Management** involves statistics-related costs and network resources' utilization. Finally, **Security Management** covers the security aspects of the network, such as physical security, access control and authorization.

The entities comprising a network management system are found in different locations in the network. The manager usually resides on the network administrator's side or on a server accessible to all administrators, whereas agents reside on the network resources being managed. The interaction between managers and agents takes place using a network management protocol such as the Simple Network Management Protocol (SNMP) and the Common Management Information Protocol (CMIP) [7].

2.1.1 Network Management Operation

In this section, the basic operations that take place between network management elements are described.

A manager is configured to monitor a set of network elements (managed objects) such as workstations, routers, switches or servers. The manager can also do an automatic topology discovery to generate a network map if properly equipped. The manager constantly polls the managed objects for specific parameters and store them in a database. Some of the polled

parameters are compared against some threshold to decide whether a problem has occurred in the network. The Manager can also send some requests to configure certain parameters on the agent, e.g. setting IP, hostname, threshold values, or initializing certain operation on the agent.

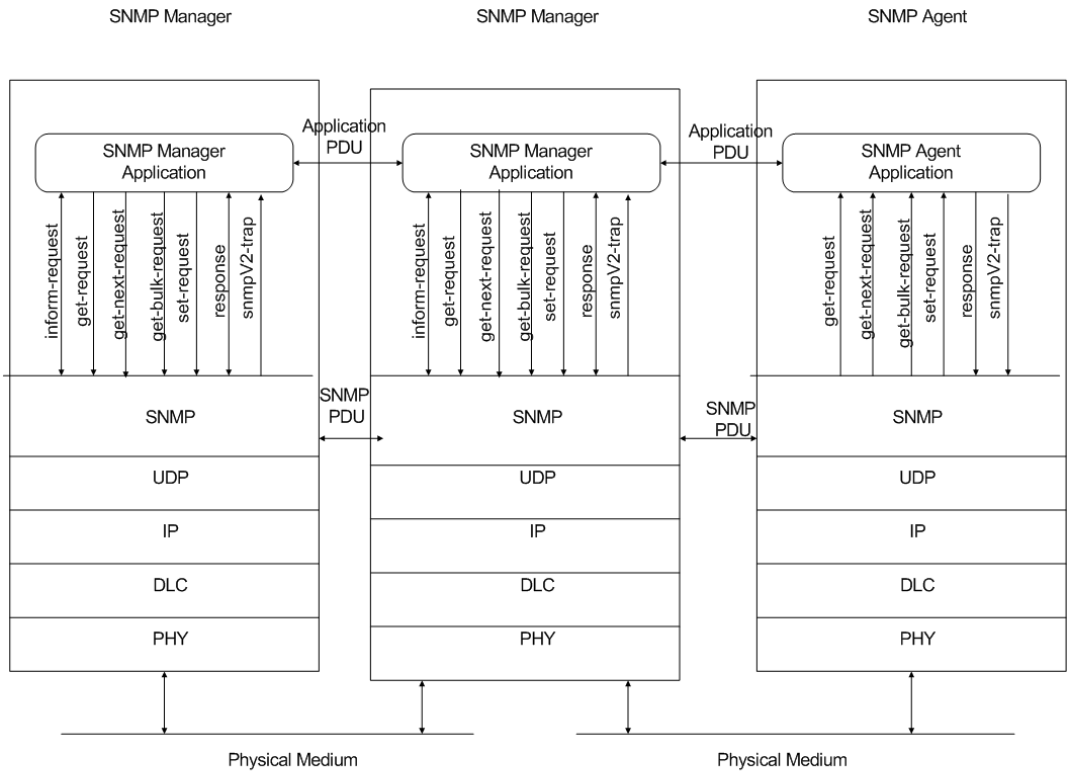


Figure 6.2 SNMPv2 Network Management Architecture

Figure 2.4: SNMP Architecture [1].

An agent on the other hand responds to manager requests for supplying or setting the required values. The agent can monitor the element's parameters and send traps to the manager when a specific threshold is exceeded.

Figure 2.4 shows the communication between managers and between a manager and an agent in the context of SNMP. For example, a *Get-Request* is sent by the manager asking for the value of a specific parameter. The answer from the agent will be a *Get-Response* carry-

Table 2.1: A Summary SNMPv2 Messages

Message	Direction	Meaning
Get-Request	M → A	Requesting the retrieval of a single value
Get-Next-Request	M → A	Requesting the retrieval of a single value right after last requested one
Get-Bulk-Request	M → A	Requesting the retrieval of large amount of data
Response	A → M M → M	Carries back requested values or acknowledging a set-request or an inform-request
Set-Request	M → A	Requesting setting the value of certain parameter
SNMPV2-Trap	A → M	Informing the manager of a notification
Inform-Request	M → M	Sent between managers for interoperablity

ing the value(s) requested. SNMPv2 has added some more messages such as the *Get-Bulk-Request* and the corresponding *Response* for retrieving a chunk of values and the *Inform-Request* for manager-manager interaction. A summary of SNMPv2 messages can be found in Table 2.1. To avoid any confusion, SNMPv2 trap messages have been enhanced and named *SNMPv2-Trap*. Similarly, *Get-Response* has been renamed *Response*.

From the description above, we can see that such behavior will increase the traffic tremendously in the network. To remedy this, SNMP has introduced Remote Monitoring (RMON) which is responsible of polling the monitored parameters and reporting a summary to the manager. When more than one RMON is used, the manager is relieved from the polling and processing of information. In addition, management traffic is reduced. Figure 2.5 represents the RMON concept.

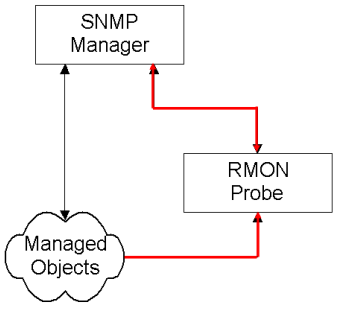


Figure 2.5: Communication Model with RMON Probe [1].

2.1.2 Network Management Approaches

In [8][9], network management approaches have been classified into four classes:

1. Centralized Network Management
2. Weakly Distributed Network Management (Hierarchical)
3. Strongly Distributed Network Management (Distributed)
4. Cooperative Network Management

The traditional **Centralized Network Management** approach is based on a single manager that interacts with multiple agents (see Figure 2.6.a). Due to the centralization of the processing in one node in the traditional approach, many concerns come into the picture such as scalability, reliability and performance. These concerns were the main drivers for finding other paradigms for network management [10]. Centralized network management is the most widely deployed and supported paradigm in today's networks [11].

In the **Weakly Distributed Network Management**, also called Hierarchical Network Management [7], Mid-Level Managers (MLM) are used between the manager and the agents (see Figure 2.6.b). MLMs take some of the responsibilities of the manager. Each MLM is responsible for a subset of agents. MLMs process the information received from agents and report it to the manager. In addition, the manager interacts with the agents through MLMs, which provide a potential for higher scalability and performance [12]. An example of this approach is RMON [13].

In the **Strongly Distributed Network Management** however, more than one manager are used where each manager is responsible for a subset of agents. In order for managers to coordinate, they may need to exchange some information (see Figure 2.6.c). Management information is also distributed or replicated on each manager. Examples of this paradigm

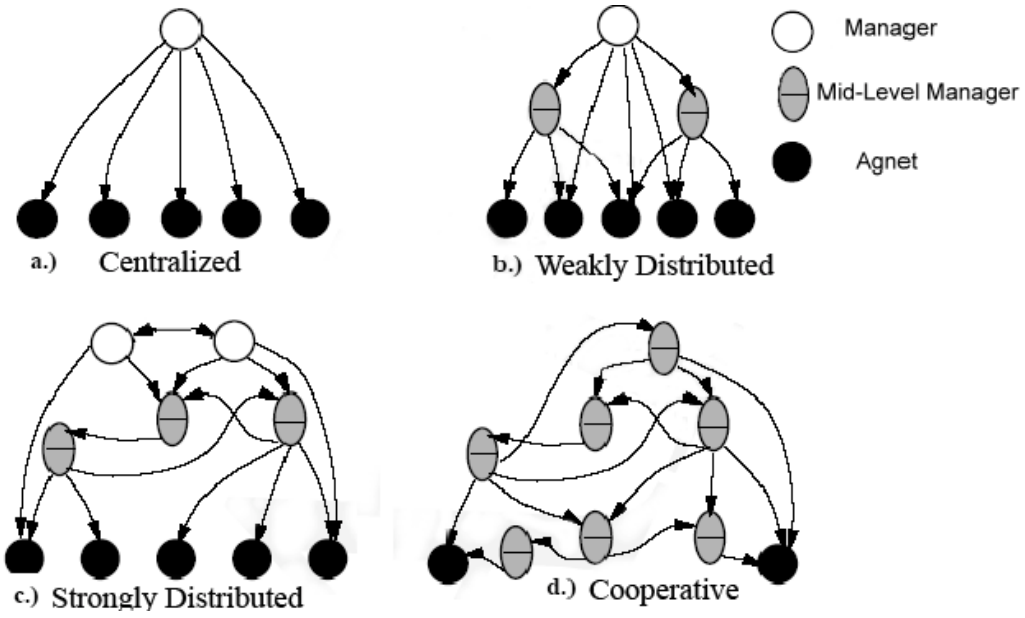


Figure 2.6: Network Management Paradigms.

include Mobile Agents (MA) [14], Management by Delegation (MbD) [15] and distributed objects [16][17].

In the **Cooperative Network Management**, MLMs and agents can be more intelligent. Being intelligent, an agent can do more jobs without consulting a manager or an MLM. An agent can talk directly to other agents or to a manager as in (Figure 2.6.d). An example of this are Intelligent Agents (IAgs) [18].

No single approach is adequate for all situations [19]. Since priorities differ from an organization to another, i.e. cost, performance, scalability, availability, etc, each paradigm might be suitable for a certain scenario.

2.1.3 Remote Operation (Remop)

As part of the continous effort of the Internet Engineering Task Force (IETF) to standardize its work in Distributed Management (Disman), Remote Operation (Remop) RFC 2925 [20] has been defined to allow management stations to request certain operations to be done by

agents or MLMs.

In the Internet draft RFC 2925, three operations were defined: ping, traceroute and lookup. The standard allows the management station to request a managed node (agent or MLM) to perform certain operations on its behalf and report the results back to it when finished. Figure 2.7 shows how this works.

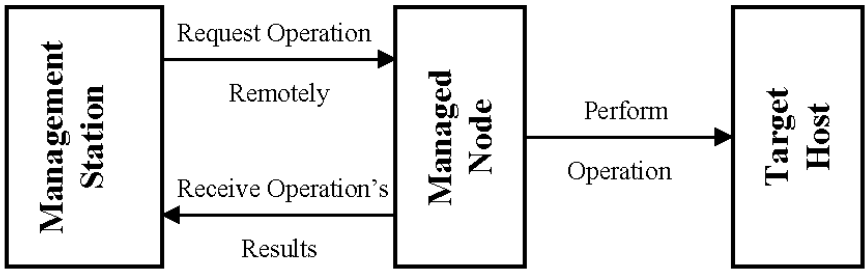


Figure 2.7: Remote Operation Model.

The draft also defines three MIBs to be used with the remop operation: DISMAN-PING-MIB, DISMAN-TRACEROUTE-MIB and DISMAN-LOOKUP-MIB. We will discuss the first MIB since it is used for testing the proposed framework. For more details about the other MIBs, the reader is referred to read [20].

The following items in the DISMAN-PING-MIB are important for the discussion:

1. `pingMIB.pingObjects.pingMaxConcurrentRequests`
2. `pingMIB.pingObjects.pingCtlTable`
3. `pingMIB.pingObjects.pingResultsTable`
4. `pingMIB.pingObjects.pingProbeHistoryTable`
5. `pingMIB.pingNotifications.pingTestCompleted`
6. `pingNotifications.pingProbeFailed`
7. `pingMIB.pingNotifications.pingTestFailed`

pingMaxConcurrentRequests

It defines the maximum number of concurrent active ping requests that are allowed within an agent implementation. A value of 0 for this object implies that there is no limit for the number of concurrent active requests in effect.

pingCtlTable

It defines the ping Control Table for providing, via SNMP, the capability of performing ping operations at a remote host. The results of these operations are stored in the `pingResultsTable` and the `pingProbeHistoryTable`.

pingResultsTable

Defines the Ping Results Table for providing the capability of performing ping operations at a remote host. An entry is added to the `pingResultsTable` when a `pingCtlEntry` is started by successful transition of its `pingCtlAdminStatus` object to enabled(1). An entry is removed from the `pingResultsTable` when its corresponding `pingCtlEntry` is deleted.

pingProbeHistoryTable

Defines a table for storing the results of a ping operation. Entries in this table are limited by the value of the corresponding `pingCtlMaxRows` object. An entry in this table is created when the result of a ping probe is determined. The initial two instance identifier index values identify the `pingCtlEntry` that a probe result (`pingProbeHistoryEntry`) belongs to. An entry is removed from this table when its corresponding `pingCtlEntry` is deleted. An implementation of this MIB will remove the oldest entry in the `pingProbeHistoryTable` to allow the addition of a new entry once the number of rows in the `pingProbeHistoryTable` reaches the value specified by `pingCtlMaxRows`.

pingTestCompleted

Generated at the completion of a ping test when the corresponding `pingCtlTrapGeneration` object is set to `testCompletion`(4). The notification contains the following values:

1. `pingCtlTargetAddressType`
2. `pingCtlTargetAddress`
3. `pingResultsOperStatus`
4. `pingResultsIpTargetAddressType`
5. `pingResultsIpTargetAddress`
6. `pingResultsMinRtt`
7. `pingResultsMaxRtt`
8. `pingResultsAverageRtt`
9. `pingResultsProbeResponses`
10. `pingResultsSentProbes`
11. `pingResultsRttSumOfSquares`
12. `pingResultsLastGoodProbe`

pingProbeFailed

Generated when a probe failure is detected when the corresponding `pingCtlTrapGeneration` object is set to `probeFailure`(0) subject to the value of `pingCtlTrapProbeFailureFilter`. The object `pingCtlTrapProbeFailureFilter` can be used to specify the number of successive probe failures that are required before this notification can be generated.

pingTestFailed

Generated when a ping test is determined to have failed when the corresponding `pingCtlTrapGeneration` object is set to `testFailure`(1). In this instance `pingCtlTrapTestFailureFilter` should specify the number of probes in a test required to have failed in order to consider the test as failed.

An operation is started by initializing a new entry in the `pingCtlTable`. The entry contains minimally the `pingCtlTargetAddressType` and `pingCtlTargetAddress` which specifies the target node to ping. After that, the test can be started either by setting the `pingCtlRowStatus` to active(1) and the `pingCtlAdminStatus` to enabled(1) or by setting the `pingCtlAdminStatus` to enabled(1) and the `pingCtlRowStatus` to `createAndGo`(4).

When the ping operation is completed, the results are stored in the corresponding entry in the `pingResultsTable` and the `pingProbeHistoryTable`. A notification can also be sent to the requesting management station with the results.

2.1.4 Network Management Enabling Technologies

As an application, network management depends on the progress of other related areas such as computer sciences and networking. In [21], many such technologies has been surveyed. Among those technologies:

- Policy-Based Network Management
- Distributed Object Computing (DOC)
- Web-Based Network Management
- Java-Based Network Management

- Code Mobility for Network Management
- Intelligent Agents
- Active Networks
- Economic Theory

In addition to the above technologies, we believe that high-availability technology has a strong potential to the network management community. High-Availability technology is very much mature and many ideas can be brought to network management specially those addressing reliability, performance and scalability.

In this thesis we build a reliable and fault-tolerant network management system. Similar concepts have been successfully applied in the clustering technology used to build high-availability servers. We introduce some fundamental high-availability and clustering concepts in the next section and then we discuss the related work in the area of fault tolerant network management system.

Chapter 3

Fault-Tolerance & High-Availability

3.1 High-Availability Technology

High-Availability (HA) is a powerful technology used to build fault-tolerant server clusters used for high performance computing or 24X7 web services. Similar concepts are used with the Hot-Standby Router Protocol (HSRP). In this section, we present HA concepts and show how they can be used to build a cluster of servers. At the end of the section, we will present an overview of a very famous HA tool called *heartbeat* that is developed under the umbrella of the Linux High-Availability Project.

3.1.1 What is High-Availability?

It is important for enterprises to have a continuous service to their clients as well as to their employees since a single wire failure may cause a service outage that may, in turn, cost a lot of money or loss of reputation. The HA concept tries to solve this problem by adding redundancy in resources in addition to automating the failover process.

Providing HA can be achieved in a variety of ways ranging from using custom redundant hardware to software-based solutions that utilize of-the-shelf hardware. The software-based

approach is preferred because it is less expensive and can be implemented using affordable customary hardware. HA found its way into many applications including building server clusters for web and enterprise services.

HA is sometimes confused with Continuous-Availability. *Continuous-Availability* (CA) refers to non-stopping services which represents the ideal case with zero downtime or never-failing services. HA on the other hand, accepts a small downtime which is the case with any real system. Note that HA does not imply CA. The downtime can be planned (e.g. for maintenance) or unplanned (e.g. a hardware failure or a disaster).

A *fault-tolerant* system refers to a system that has the ability to continue working despite a hardware or a software failure. One way to achieve fault-tolerance is by avoiding a *Single Point of Failure* (SPOF) that results from components with no redundancy and whose loss results in a loss of service. The process by which a redundant component resumes the service after a failure is called *Failover* [22].

HA has been in existence for quite some time and many research projects have been investigating and developing new techniques and tools to provide HA. Linux High-Availability Project [23], Linux Virtual Server (LVS) [24], the Horus Project [25] and the ISIS Project [26] are among the most famous ones. There also exist many commercial HA products from big companies such as IBM, Compaq, HP, Microsoft, Novel and others. To coordinate and standardize those efforts, the IEEE has formed a task force called IEEE Task Force on Cluster Computing (IEEETFCC) [27].

3.1.2 High-Availability Server Clustering Problem

HA can be better understood using one of its applications. The application we are discussing "building a cluster of servers" will also illustrate the ideas and concepts behind having HA.

The problem of building a cluster of servers is an interesting problem. The cluster we are building must have the following properties:

1. Appears to the user as one server.

2. Provides 24x7 availability through a fault tolerant design.

3. Scalable.

4. Makes efficient use of the resources it includes.

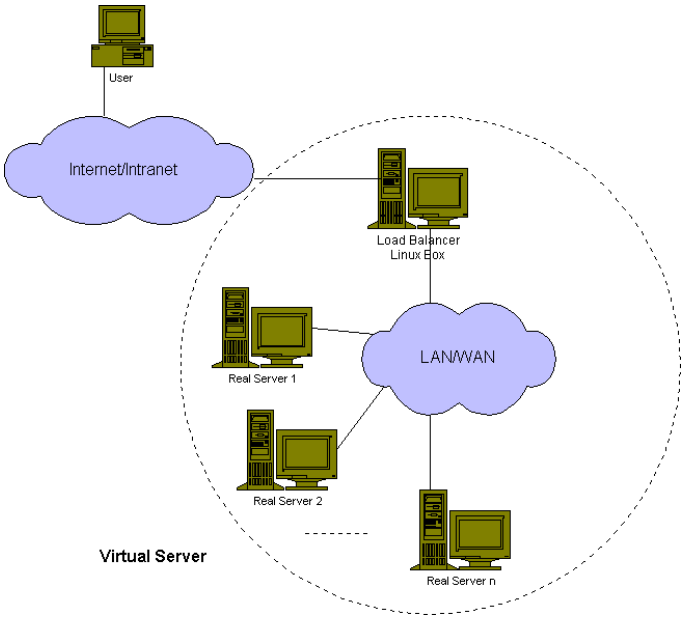


Figure 3.1: Virtual Server Model.

The first property can be achieved by using a single server that receives clients' requests and forwards them to the appropriate server. The server that receives the requests is called a *load balancer* and the servers that do the actual processing are called *real servers*. The overall system is called a *virtual server*. Figure 3.1 shows a model of a Virtual Server. In the following section we discuss some approaches used to design a virtual server. We will discuss the virtual server via Network Address Translation (NAT), the virtual server via IP Tunneling and the virtual server via Direct Routing.

The second and third properties might seem conflicting with the first one for a while, but actually they are not. Since having one node to receive clients' requests will require only one

IP address, this will result in a single point of failure (SPOF). Using a single load balancer will also limit the scalability of the cluster. On the other hand, using more than one node to avoid SPOF will result in different addresses used for each server. There are however ways to overcome that, e.g. using Virtual IP.

The last property means mainly having all real servers utilized and avoiding occasions when one server is overloaded while other servers are under-loaded. This can be overcome by load-balancing the requests among servers using an appropriate scheduling algorithm. We will discuss a set of scheduling algorithms including: Round-Robin Scheduling, Weighted Round-Robin Scheduling, Least-Connection Scheduling, Weighted Least-Connection Scheduling, Locality-Based Least-Connection Scheduling, Locality-Based Least-Connection with Replication Scheduling, Destination Hash Scheduling and Source Hash Scheduling.

3.1.3 How Does High-Availability Work?

The basic concept behind HA tools is to allow the involved nodes to periodically exchange small messages called *keepalive messages* to indicate that the node is still active. Whenever a node stops sending keepalive messages, its partner can assume that it is dead and can continue serving on its behalf.

In addition to sending, receiving and processing keepalive messages, HA manages the failover process between resources. If only two nodes are involved, they can be working in one of two modes: active/passive or active/active. If more than two nodes are involved, then they can have any of the following modes: N+M, N, cascading and multidirectional backup modes. The failover can also be automatic or manual. More details about those configuration will be given in the coming section.

The process is not simple as it seems since many issues have to be addressed by HA tools:

- The amount of traffic generated by the keepalive messages must not consume a lot of bandwidth.

- The messages must not be sent at long intervals to allow for fast failure detection.
- Security is another important issue, since the listening node must be able to authenticate the messages received.
- If more than one node is involved, then there has to be an election mechanism as which node should takeover the failed node.
- The data on the failed node must be replicated elsewhere to allow the partners to use it.
- The failure detection mechanism should also have the ability to distinguish between different types of failure. For example, when the hard drive crashes on the active node, the keepalive messages may not stop which will not allow the backup node to detect the failure. Another example is the *split-brain* problem where due to a network problem for instance, each node thinks that the other is dead but it is not. The latter must be avoided because it may result in sever problems when the two nodes write to a shared data simultaneously.

3.1.4 Failover Configurations

This subsection deals with configurations that allow fault tolerant operations [28]. As we discussed before, using one load balancer will result in a SPOF, and hence we must use more than one load balancer to guarantee an uninterrupted service.

In the **active/passive configuration**, a secondary load balancer is on a hot-standby backup for the primary one. When the primary load balancer fails, the secondary load balancer takes over and starts serving the coming requests.

The secondary load balancer takes over the identity of the primary one through the use of a virtual IP address.This approach is used in the Linux Virtual Server (LVS). Linux allows one interface to have a second virtual IP address in addition to the real IP it has. When the

secondary load balancer is up, it uses the virtual IP of the first one and uses ARP spoofing [29] to change the mapping that exists in the clients and switches caches to map to its physical MAC address [30].

Load balancers keep track of the status of each other through the use of heartbeats that are sent between the two nodes periodically. When one node stops hearing the heartbeats from the other one, it assumes that the other node failed. To improve the dependability of the heartbeats mechanism, more than one channel are used for heartbeats. One channel can use a serial cable between the two nodes and the other channel uses an Ethernet interface.

In the **active/active configuration** both load balancers work simultaneously. When one of the load balancers fails, the other one takes over the primary as in the previous case. The **N** and **N+M configurations** where N is the number of active nodes and M is the number of passive nodes are just extensions of the active/passive and active/active cases but with more than two nodes.

The other configurations refer to the way applications fail, not with the clustering nodes. In the **cascading configuration** each application or service has a priority associated with it and the clustering node. The application or service starts on the server with the highest priority for that application. When this node fails, the application tries to start on the node with the second highest priority. In all cases the switching upon failure can be automatic or manual. If **automatic**, no human intervention is needed. Otherwise, an administrator must do the switching **manually**.

3.1.5 Clustering Techniques

In this section we describe some techniques that enable a cluster of severs to appear as one single server from the client's side [31].

Virtual Server via NAT: NAT is a popular technique used to hide the internal addresses used in LANs from the public Internet. NAT techniques can also be used to build a

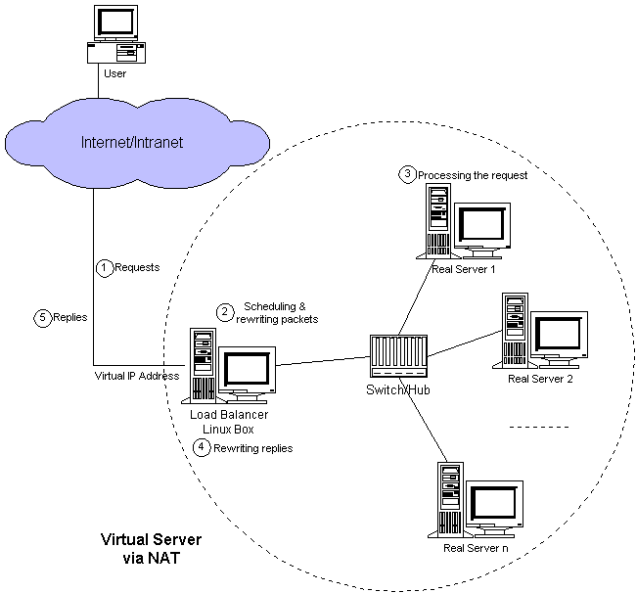


Figure 3.2: Virtual Server via NAT.

cluster of servers. In this approach, the load balancer has a single IP address, called the virtual IP address, that is known everywhere. The real servers on the other hand have private addresses hidden from the clients. The architecture is depicted in Figure 3.2. The load balancer receives the packets sent by the clients and changes the destination IP address to that of the real server and forwards the packet to the real server. The response is then sent by the real server to the load balancer which in turns changes the source IP to its own IP and forwards the response back to the client. The load balancer keeps track of the active connections using a hash table to forward all packets belonging to the same connection to the appropriate real server.

The advantage of this architecture is that only one IP address is needed. The load balancer can run any OS that supports TCP/IP suite. On the negative side however, the load balancer may become a bottleneck since it handles both requests and responses and hence reduces scalability. Another limitation is that real servers must be within the same LAN.

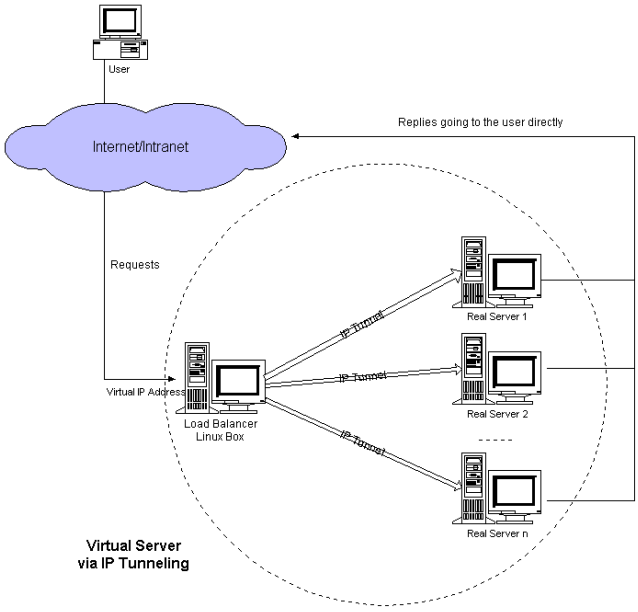


Figure 3.3: Virtual Server via IP Tunneling.

Virtual Server via IP Tunneling: This situation is depicted in Figure 3.3. In this approach the load balancer receives the clients' requests, encapsulates them inside another IP packet and sends them to the real server. The real server in turn decapsulates the packet, processes it and sends the response directly to the client. The procedure is depicted in Figure 3.4.

The advantage of this approach is its great potential for scalability that results from the fact that requests handled by the load balancer are very small compared to the responses sent by real servers. Another advantage is that servers can be geographically distributed anywhere in the Internet. The disadvantage is the need for OS with IP tunneling support. This problem is not that serious as IP tunneling is becoming a standard in all networking OS. Another problem results from the tunneling overhead.

Virtual Server via Direct Routing: For direct routing to work, the real servers and the load balancer must be located on the same LAN segment as shown in Figure 3.5. When the load balancer accepts an IP packet, it forwards it to the appropriate real server using the

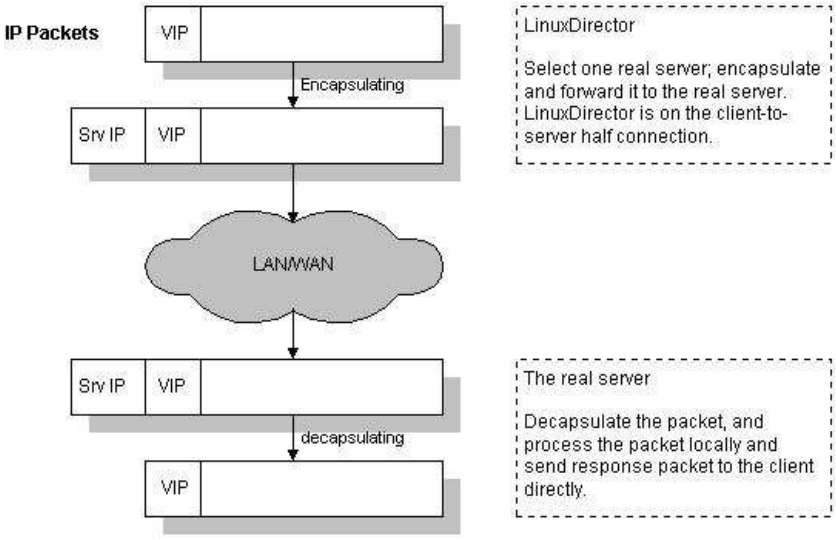


Figure 3.4: IP Tunneling Process.

real server physical Media Access Control (MAC) address. The real server can then send the response directly to the client. The advantage of this approach is scalability.

To be able to move TCP connections from the load balancer to real servers, both the load balancer and real servers must have the VIP assigned to them. For this to work, the real servers' interfaces must not send an Address Resolution Protocol (ARP) response [32]; otherwise whenever an ARP request is sent for the VIP, the real servers and the load balancer will reply to the ARP request each with its MAC address.

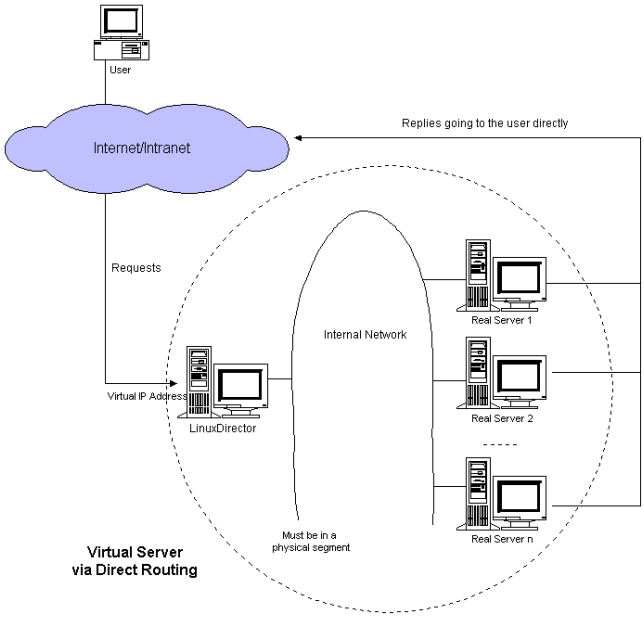


Figure 3.5: Virtual Server via Direct Routing.

3.1.6 Scheduling Algorithms

Many scheduling techniques have been introduced in the literature. Each has its pros and cons and each might be applicable to a different scenario. These techniques can be used to load balance requests received from clients among servers [23][33]. The following is a brief description of some techniques:

Round-Robin Scheduling (RR): RR scheduling distributes jobs among servers in a sequential manner. One implementation of this is the Round-Robin Domain Name Server (RR-DNS) which uses an extension of the DNS protocol to map requests destined to the same host name to different IP addresses and hence to different real servers.

The RR treats all servers as having the same computing power. It also assumes that all jobs have the same processing time. Hence if one of these assumptions is not valid - which is the case in most if not all Internet servers- an imbalance situation will occur. The RR-DNS also has drawbacks over other RR implementations that result from DNS caching. This problem occurs when the client caches the IP of the last request and hence keeps using the same IP. This will indeed result in an imbalance situation.

Weighted Round-Robin Scheduling (WRR): This approach differs from the RR by assigning different weights to different real servers. This weight indicates the server computing power. This algorithm assigns jobs to servers in a Round-Robin fashion but assigns more jobs to servers with higher weight. For example, if server1 is twice as powerful as server2, then server1 will receive two jobs for each job received by server2.

Least-Connection Scheduling: This approach keeps track of the number of active connections on each real server and gives more jobs to servers with fewer active connections. This approach is adaptive or dynamic and hence more complex but can do a better job than the previous two. The drawback of this approach is that it assumes the same processing power for all real servers.

Weighted Least-Connection Scheduling: This approach modifies the previous one by considering the processing power of real servers. Therefore, jobs are distributed based on the number of active connections relative to their processing power.

Locality-Based Least-Connection Scheduling: This approach is designed to be used with proxy-cache server clusters. It considers the IP of the server and forwards the request to that server unless the number of connections on that server is above its capacity. In the later case, jobs will be assigned to a different server.

Locality-Based Least Connection with Replication Scheduling: This approach is also used with proxy-cache server clusters. It differs from the Locality-Based Least-Connection in that it assigns jobs destined to certain IP to a subset of real servers. The job is then assigned to the server with lower number of active connections within this subset. If all servers of that subset are above their capacity, the real server with least number of connections from the overall cluster is added to the subset, and the most heavily loaded server from the subset is dropped from the subset.

Destination Hash Scheduling: This approach distributes jobs among servers based on their destination IP. It uses a hash table to assign jobs to servers.

Source Hash Scheduling: This approach distributes jobs among servers based on their source IP. Similar to the Destination Hash Scheduling, it uses a hash table to map jobs to real servers.

3.1.7 Linux High-Availability Project

The Linux-HA project is a very successful project that has many contributors from the academic and industrial sectors. The project is good in that many other projects have added tools that can be easily integrated with it. The main project is around the heartbeat program for

Linux-HA. However, other partner projects such as the LVS and DRBD are useful and will be introduced in this section.

The Heartbeat Application

The heartbeat application is the central and the most well-known component of the Linux HA project. The heartbeat program sends heartbeats between the nodes of the HA system. The program also monitors and controls the various shared resources on each node.

The story behind the project started when Harald Milz started a mailing list in 1997 to discuss how to provide HA to Linux. As a result of the discussion, Harald wrote a howto about how one can provide Linux with HA. Later, Alan Robertson found that howto and used it to write the code of the heartbeat program. The first version of the heartbeat was launched in 1999 [34].

As a software, Linux HA project consists of many components. Each component is a single program that does certain tasks. This modularity enhances the scalability of the heartbeat and eases the integration with other tools. Figure 3.6 shows how the different daemons of the heartbeat program communicate. Note that no resources are active on the backup node except for the heartbeat and the *send_arp*. The other resources get activated by heartbeat when changing from passive to active.

When the heartbeat daemon on the backup LD stops hearing heartbeats from the active LD, it instructs the *send_arp* daemon to assign the IP address of the active LD to the interface of the backup LD. It does that using ARP Spoofing [29] where it sends ARP replies to all real servers to change the IP/MAC mapping to its own MAC address. The *send_arp* daemon also sends a request to the active LD to shutdown the *ldirectord* daemon on the active LD. The heartbeat starts the *ldirecord* deamon on the backup LD and starts any other resources needed. Table3.1 shows a summary of tasks done by each component of the heartbeat and LVS.

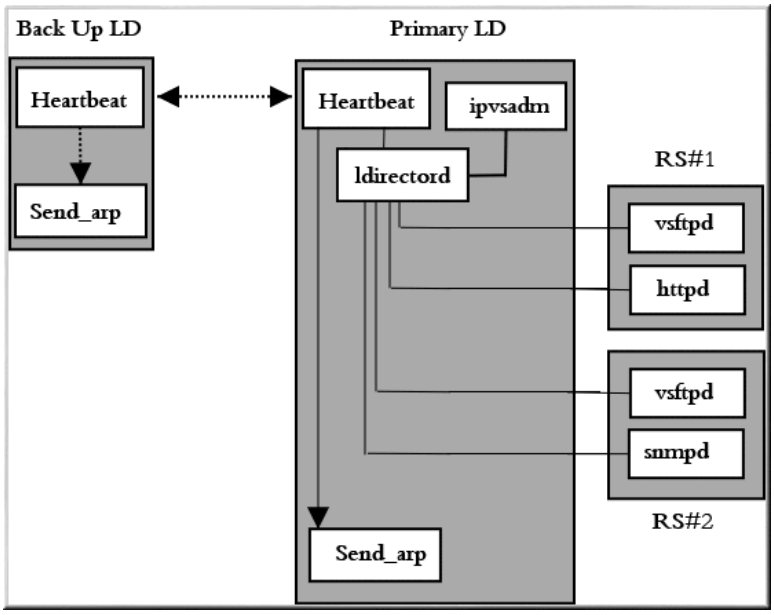


Figure 3.6: The Interaction Between Heartbeat & LVS.

The heartbeat defines two failover modes: active/passive and active/active. It is important to note that the active/active mode provided by heartbeat is not a full active/active mode. We will explain why in this this section.

In the **active/passive** mode, only one node can run the shared resources. Those resources may include Virtual IP (VIP), web server (e.g. httpd), mounting disk, etc. When the active node fails, the heartbeat daemon on the passive node starts the resources and continues

Table 3.1: Task of Different Heartbeat & LVS Components.

	Daemon	**Tasks**
1	*heartbeat*	- Starts the *ldirectord* daemon - Sends heartbeats to the active LD - Responds to heartbeats from backup LD
2	*ldirectord*	- Calls the *ipvsadm* to configure routing table - Monitors services on RS
3	*ipvsadm*	- Maintains the routing table
4	*send_arb*	- Claims the IP of the active LD on failure - Sends a request to shutdown the *ldirectord* on the active LD and starts the *ldirectord* on the backup LD

working.

In the **active/active** mode however, both nodes can have active resources running simultaneously but they have to be disjoint. For instance, each node has a different IP, different disk, etc. This is useful when we have two servers running different services but we want them to work together and be a backup for each other. This is not the real active/active mode but it can be useful sometimes. In our implementation, this mode is used in the middle layer (Mid-Level-Manager).

There exists another project called SARU [35] which tries to modify the heartbeat program to support active/active configuration. But, their solution is incomplete as of this writing. The idea of SARU is to assign the two nodes the same IP and the same MAC addresse. Of course, by doing that each packet will be received by both nodes. In order not to process each request twice, packets can be filtered based on some rules. Packet filtering can be static or dynamic. Static filtering can be based on odd/even source IP addresses for example. Whereas dynamic filtering can be more involved and requires communication between the two nodes. In Linux, such filtering can be done by changing the configurations of the *iptable* module.

The heartbeat program can send the heartbeats in many ways. It supports UDP unicast, UDP multicast, UDP broadcast and serial communication.

When installing the heartbeat program, all configuration files will be stored in the directory /etc/ha.d/. Three configuration files are essential for heartbeat to operate correctly, namely *ha.cf, haresources and authkeys*. Additionally, a file called *ldirectord.cf* which can be found in /etc/ha.d/conf is used to integrate LVS with the heartbeat by defining the real servers, scheduling , etc. Table 3.2 summarizes the configuration files and their use.

The heartbeat supports one more feature which is “auto failback”. This feature allows the failed node to restore its primary role after it becomes active again. In other words, if the “auto failback” is disabled and the failed node is up again, it stays passive till the current

Table 3.2: The Heartbeat Configuration Files.

File	Purpose
ha.cf	Specifies the heartbeat communication medium, rate, protocol and nodes involved. It also specifies add-ons if they exist.
haresources	Defines resources to start and stop on each node.
authkeys	Defines encryption protocol used.
ldirectord.cf	Specifies the real servers, services, scheduling.

active node (which used to be passive) fails. However, if the "auto failback" is enabled and the failed node is up, it becomes active and the current active node returns to passive. This is useful when we have one node that is more powerful than the other and we would like it to be the primary whenever possible. Enabling and disabling "auto failback" can be done by setting the *auto_failback* option in *ha.cf* file to on or off. see Table 3.3.

Table 3.3: ha.cf Configuration File's Directives.

Directive	Default Value	Meaning
debugfile	/var/log/ha-debug	File to write debug messages to
logfile	/var/log/ha-log	File to write other messages to
logfacility	local0	Facility to use for syslog()/logger
keepalive	2	Time between two consecutive heartbeats in seconds
deadtime	30	# of seconds to declare a host as dead
warntime	10	Time in seconds before issuing "late heartbeat" warning in the logs
initdead	120	Similar to the deadtime but with network delay after a reboot
udpport	694	UDP port to use for bcast/ucast communication
bcast	eth0	Interface to use for broadcast
ucast	eth1 172.16.70.153	Interface and node for unicast
auto_failback	on/off	Activates & deactivates active-active mode
watchdog	/dev/watchdog	Reboot the machine after a minute of being sick
node	lvs-ld1	The nodes involved in the heartbeats

Table 3.3 summarizes the main configuration directives for the heartbeat operation. The *keepalive* defines the interval for sending heartbeats in seconds. The *deadtime* defines the length of the period through which, if no heartbeat is received, the corresponding node is declared as dead. With some configurations, the network takes some time to start working after a reboot. This is handled by the *initdead* directive which should be at least twice the normal *deadtime*.

The heartbeat has many graphical configuration tools among the most popular ones are

the *Ultra Monkey* [36], *Keepalived* [37] and *Piranha* [38].

STONITH

When a node in the cluster appears to be dead, it is not a certainty that it really is. The reason is that the disappearance of the heartbeats can be due to many reasons other that the complete failure of the machine (a complete power off). An example could be when a software failure happens, heartbeats stop, but the node is still active and using the shared disk. In such a case, there has to be a way for the HA tool to shutdown the machine to make the anticipation a reality.

STONITH (Shoot The Other Node In The Head) is a node fencing mechanism that allows powering down or reboot a remote machine. When doing that, we can be certain that only one node is active at a time. STONITH is part of Linux HA and it can be configured through the *ha.cf* file.

ipfail

Ipfail is an add-on to the heartbeat written by Kevin Dwyer. The idea behind ipfail is to account for the situation where the failover must be started due to a failure on the network rather than a failure on the active node.

For example, suppose that we have two nodes, one active and one passive. Although not common, it is possible to have a situation where the client can reach the passive node but not the active one. In such a situation the default settings of the heartbeat will not switch the two nodes. However, *ipfail* can force the failover to the passive node.

The operation of the *ipfail* depends on pinging certain node(s) in the network which reflects the connectivity of the intended clients. Whenever the node is not reachable, the *ipfail* daemon on the active node communicates with the *ipfail* daemon on the passive node to check the connectivity. If the connectivity of the passive node happens to be better, a failover is ini-

tiated. Of course the operation of the *ipfail* depends heavily on the selection of the nodes to ping which must be studied carefully.

Softdog

Softdog is not part of the Linux HA package but rather a watchdog implementation which comes as a part of Linux kernel. The *softdog* reboots the local system whenever it becomes unstable or sick. This is extremely useful to avoid situations when a server malfunctions but does not stop sending heartbeats. In such a situation, it becomes tricky for the passive node to discover the failure. The *softdog* can be used by *heartbeat* through the *watchdog* directive.

Linux Virtual Server (LVS)

A Linux Virtual Server (LVS) is a High-Availability (HA) solution that implements most of the concepts discussed earlier [24][31]. LVS can be easily integrated with heartbeat to implement a full virtual server.

The architecture of the LVS is shown in Figure 3.7. The system consists of a front-end consisting of load balancers (usually two) called LinuxDirectors. The job of the front-end is load balancing through the use of any of the scheduling algorithms discussed in the previous subsection. The system can also be used with any clustering mechanism discussed earlier. The front-end also implements an active/passive failover configuration. Unfortunately, most Linux distributions do not have support for other failover configurations [39].

The second part of the system consists of a pool of real servers. These servers do the actual processing of jobs received from the load balancers. The system also comprises a back-end storage that can be shared or distributed between real servers.

The LVS software must be installed along with the heartbeat software on the LDs. By proper configuration, LVS controls the forwarding of requests from the LD to real servers. LVS also implements the scheduling algorithm specified.

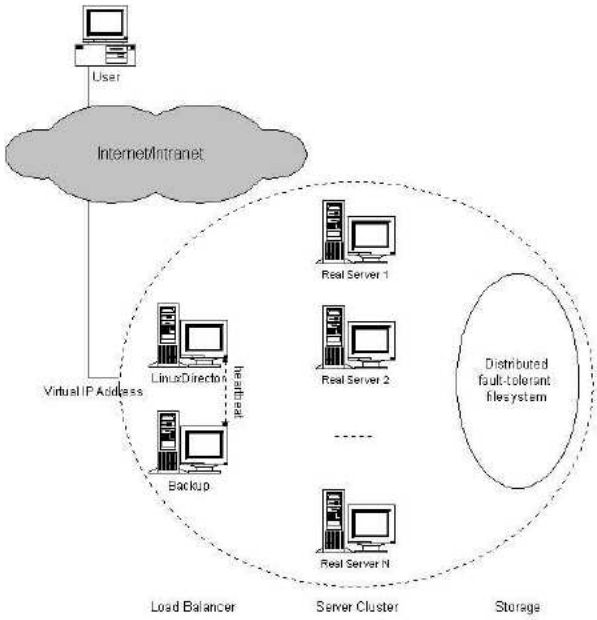


Figure 3.7: Linux Virtual Server.

Two daemons are responsible for the LVS operation: *ldirectord* and *ipvasdm*. The *ldirecord* is part of the heartbeat package while *ipvsadm* is a part of the LVS project. Figure 3.6 shows how these components fit with the heartbeat daemon.

When the *ldirectord* daemon is started, it calls the *ipvsadm* service to configure the routing table in the kernel and starts monitoring the services on the real server. When a service is detected as malfunctioning, *ldirectord* instructs the *ipvsadm* to remove the corresponding real server from the routing table.

The *ipvsadm* maintains the routing table on the active LD. It adds, deletes and changes entries on the table based on requests from the *ldirectord* daemon.

Database Replication

An important part of any fault-tolerant system is a replicated image of the data used by the system. Many solutions exist that provide data replication. Examples of such systems include:

- SCSI Sharing
- Dual Controller RAID
- Shared Fiber-Channel Disks
- High-End Storage

The problem with these solutions is their high costs. In addition, some of them, such as SCSI sharing, have a single point of failure which is the disk itself.

An innovative solution that is a part of the Linux High Availability Architecture is the Distributed Redundant Block Device (DRBD). DRBD was written by Philipp Reisner. It consists of daemons running on several machines connected by a TCP/IP network. The daemons take care of synchronizing images of the same data on all the involved machines. The participants can be two or more machines.

Each DRBD daemon keeps monitoring the device it is responsible for. Whenever a WRITE request is sent to the disk, the request is duplicated in all other machines in the group.

Compared to other solutions, DRBD is free and avoids SPOF. However, it has a limitation, that is we can only have one primary node at a time. The primary node is the only node that can write to its shared disk. All other machines are considered as secondaries and just keep receiving the changes and writing them to their disks. When the primary fails, the heartbeat assigns another node as a primary.

On the primary node, part of the local disk is used to record all changes made to the DRBD disk, this is called metadata. Whenever a synchronization check point is reached, the changes recorded in the metadata are sent to the secondary nodes.

Let’s explain the operation of DRBD by showing the possible scenarios. To clarify the examples, we will use two servers NMS1 (primary) and NMS2 (secondary) :

- The obvious case is when the NMS1 fails. In this case, NMS2 will become the primary. When NMS1 comes back, it stays as the secondary node and a full synchronization from NMS2 to NMS1 is made, since we are not sure which blocks have been changed on NMS1 just before the failure.

- When NMS2 fails while it is the secondary, NMS1 continues recording the changes it makes in the metadata section. When NMS2 comes back, an incremental synchronization is made from NMS1 to NMS2. If for some reason NMS1 fails while NMS2 is in failure, a full synchronization is made from NMS1 to NMS2 when both are up.

3.2 Fault-Tolerance

In order to evaluate the fault tolerance capabilities of different systems, we must develop a model and define some metrics for such a model. In the following section, we define reliability, maintainability and availability. Some definitions are given in the context of networking, but they apply to other areas as well.

A **system** is a collection of components arranged according to a specific design in order to achieve desired functions with acceptable performance and reliability measures [40]. A **failure** of a system occurs when the behavior of the system first deviates from that required by its specifications. Hence, failures can be classified into two types, critical failures and non-critical failures. An example of a non-critical failure is when a server within a server cluster fails. In such a case, even though the performance of the cluster might be degraded, the system is still operational. On the other hand, if the main switch that connects the servers cluster to the network fails, the whole cluster would be isolated. The latter is an example of a critical failure.

Reliability describes the ability of a system to work correctly for a particular period. Reliability $R(t)$ is a function of time and is defined as "the conditional probability that the system

will perform correctly throughout the interval $[t_0, t]$, given that it was performing correctly at time t_0". A measure that can describe the reliability of a particular system is the **Mean Time Between Failures (MTBF)**. MTBF is the "average time between two consecutive failures".

The Failure rate (λ) is defined as:

$$\lambda = \frac{1}{MTBF} \tag{3.1}$$

The reliability of a system is proportional to its MTBF and inversely proportional to its λ. Some references differentiate between MTBF and the Mean Time between Critical Failures (MTBCF). The difference is made based on whether the failure is critical or not. In our discussion, we will not follow that convention and will be using MTBF assuming that all failures under consideration are critical.

A measure that describes the maintainability of a system is the **Mean Time to Repair (MTTR)**. MTTR is "the average time required to restore the system to an operational status once it has a failure". MTTR includes fault detection, isolation, the time to deliver the necessary parts to the location, the time needed to replace the components, testing and restoring the full service. Figure 3.8 shows a pictorial representation of MTBF and MTTR.

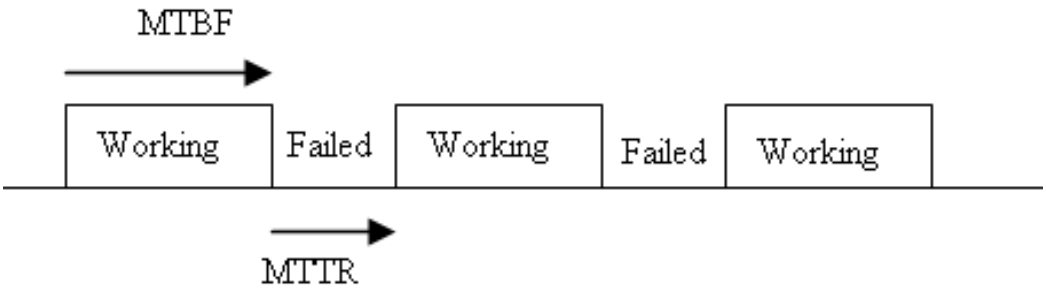


Figure 3.8: The Relationship Between MTBF and MTTR.

The repair rate (μ) is defined as,

$$\mu = \frac{1}{MTTR} \tag{3.2}$$

A third measure is availability. **Availability** describes the relationship between the frequency of critical failures and the time to restore service. Availability $A(t)$ is a function of time defined as "the probability that the system is operating correctly at an instant of time t". The difference between availability and reliability is that reliability is defined over a period of time while availability is defined at an instant of time. A system can be available yet experiences frequent periods of inoperability as long as the length of each period is short.

A distinction between reliability and availability can be seen by observing that reliability addresses only unexpected failures, while availability addresses both unexpected failures and service shutdown due to intentional maintenance [41].

From the definition above, availability can be expressed as the ratio between the time during which the system is operational and the total time. Mathematically, this can be expressed as,

$$A = \frac{MTBF}{MTBF + MTTR} = \frac{\frac{1}{\lambda}}{\frac{1}{\lambda} + \frac{1}{\mu}} = \frac{\mu}{\mu + \lambda} \tag{3.3}$$

One simple model used to model reliability is the constant-failure rate model. The model assumes - as its name indicates- that the failure rate is constant. In that model, the reliability $R(t)$ is expressed as,

$$R(t) = e^{-\lambda t} \tag{3.4}$$

The model in Figure 3.9 shows that as time goes to infinity, the reliability goes to zero which is the case of all man-made systems.

3.2.1 Reliability Estimation

We will classify systems according to their configuration, that is how their components are arranged. Thus, systems can be classified as series systems, parallel systems, series-parallel systems, parallel-series systems and complex systems.

In the following paragraphs, we are going to present a mathematical expression for the reliability of each class of systems. But before we start building the mathematical model for reliability, we need to define few terms. Assume we have a system S that consists of n units. For each unit i we define the following:

$$x_i = \text{the event that the } i\text{th unit is working,}$$
$$\bar{x}_i = \text{the event that the } i\text{th unit is in failure,}$$
$$P(x_i) = \text{the probability that the } i\text{th unit is working,}$$
$$P(\bar{x}_i) = \text{the probability that the } i\text{th unit is in failure,}$$

We also define:

$$R(S) = \text{the reliability of the system S, and}$$
$$P_f(S) = \text{the unreliability of the system S.}$$

It is clear that by definition:

$$P_f(S) = 1 - R(S)$$

Series Systems

Figure 3.10 shows an example of a **series system**. From the figure we can see that a series system has all components connected in series. A failure of any component will result in the

failure of the whole system. Hence, for a series system to be operational, all of its components must be operational. Then, from basic probability we have the following:

$$R(S) = P(x_1 x_2 ... x_n)$$

Furthermore, we can have two types of components, dependent components and independent components. *Dependent components* are the components whose reliability is affected by the failure of other components. For example, the failure of the cooling system, may increase the probability of the failure of a car's engine. *Independent components* on the other hand are not affected by the failure of other components. For example, a failure of a piece of software on a computer although may affect the operation of the system, but it might have no impact on the reliability of its hardware.

We can rewrite the reliability equation for dependent systems as

$$R(S) = P(x_1)P(\frac{x_1}{x_2})P(\frac{x_3}{x_1 x_2})....P(\frac{x_n}{x_1 x_2 x_3 ... x_{n-1}})$$

However, for independent systems it will be

$$R(S) = P(x_1)P(x_2)...P(x_n)$$

or

$$R(S) = \prod_{i=1}^{n} P(x_i)$$

It is important to note that the reliability of a series system is less than or equal to that of the least reliable component.

Parallel Systems

In a parallel system, components are connected in parallel so that a failure in one component does not interrupt the operation of the system and other backup components can always take the role of the failed one. Figure 3.11 shows an example of a parallel system. Hence, for a parallel system to fail, all the parallel components have to fail. We can write this as:

$$P_f(S) = P(\bar{x}_1\bar{x}_2....\bar{x}_n)$$

Since $R(S) = 1 - P_f(S)$, then for the case of dependent components:

$$R(S) = 1 - P(\bar{x}_1)P(\frac{\bar{x}_1}{\bar{x}_2})P(\frac{\bar{x}_3}{\bar{x}_1\bar{x}_2})....P(\frac{\bar{x}_n}{\bar{x}_1\bar{x}_2\bar{x}_3...\bar{x}_{n-1}})$$

For independent components however, $R(S)$ can be written as:

$$R(S) = 1 - P(\bar{x}_1)P(\bar{x}_2)...P(\bar{x}_n)$$

or

$$R(S) = 1 - \prod_{i=1}^{n} P(\bar{x}_i)$$

In case the reliability $P(x_i)$ for all components is the same, we can write $R(S)$ as:

$$R(S) = 1 - (1 - p)^n$$

where p is equal to $P(x_i)$.

We can also observe that the reliability of parallel systems is greater than the reliability of the most reliable component in the system.

Parallel-Series Systems

A parallel-series system is made out of m parallel paths, each path contains n components connected in series. Figure 3.12 shows an example of a parallel-series system. We will use $(i = 1, 2, ..., m)$ to index paths and $(j = 1, 2, ..., n)$ to index serial components within a path. Similarly, $P(x_{ij})$ will refer to the reliability of the jth component in the ith path.

Since each parallel path is made of serial components, it is a must for all components of a path to operate in order for the path to operate. Then, the reliability of path i will be:

$$P_i = \prod_{j=1}^{n} P(x_{ij}) \qquad where \quad i = 1, 2, ..., m \quad and \quad j = 1, 2, ..., n$$

Let $\bar{P}_i$ be the unreliability of the path i, then, the reliability of the system will be:

$$R(S) = 1 - \prod_{i=1}^{m} \bar{P}_i$$

By substituting for $\bar{P}_i$,

$$R(S) = 1 - \prod_{i=1}^{m} [1 - \prod_{j=1}^{n} P(x_{ij})]$$

If the reliability of all components is equal, we can express $R(S)$ as:

$$R(S) = 1 - (1 - p^n)^m$$

Series-Parallel Systems

A series-parallel system consists of n series subsystems connected in series each of which consists of m components in parallel. A typical example can be seen in Figure 3.13.

we can express the unreliability of a subsystem i as:

$$\bar{P}_i = \prod_{j=1}^{m}(1 - P(x_{ij}))$$

The reliability of the whole system is similar to the reliability of a series system.That is

$$R(S) = \prod_{i=1}^{n} P_i$$

or

$$R(S) = \prod_{i=1}^{n}[1 - \prod_{j=1}^{m}(1 - P(x_{ij}))]$$

In case all components have the same reliability,

$$R(S) = [1 - (1 - p)^m]^n$$

Complex Systems

So far, the discussion has been limited to regular systems. In real life however, systems are mostly irregular; e.g., computer networks, cars, factory production lines, etc. In Figure 3.14, we can see an example of such systems.

Many approaches exist to deal with such systems, among which are:

1. Decomposition Method
2. Tie-Set & Cut-Set Methods
3. Event-Space Method
4. Boolean Truth Table Method
5. Reduction Method

6. Path-Tracing Method

7. Factoring Algorithm

In the next section, we will explain an example of the decomposition method which is used to evaluate our proposed system. More information about other techniques can be found in [40].

3.2.2 Decomposition Method for Evaluating Systems Reliability

The decomposition method utilizes a Reliability Block Diagram (RBD). An RBD shows the connections between different system's components from a reliability point of view. Figure 3.14 shows an example of an RBD.

In the decomposition method, the idea is to "decompos" a complex system into simpler subsystems. Then, we evaluate the reliabilities of the simpler systems, and use them to find the reliability of the complex system using some probability relations.

The technique starts by selecting one component (called module x) that appears to link the reliability structure of the system. After selecting module x, the reliability of the system can be expressed using Bay's Theorem as:

$$R(S) = P(\text{system working}|x)P(x) + P(\text{system working}|\bar{x})P(\bar{x})$$

Let's take an example based on the system shown in Figure 3.14. We will consider component B to be module x selected for decomposition. Now, we can decompose the original RBD into two RBDs based on whether B is working (Figure 3.15-a) or not (Figure 3.15-b).

The reliability of the system in Figure 3.15-a is that of a simple parallel system and it can be written as:

$$R(\text{system working}|B) = 1 - \bar{P}(D)\bar{P}(E)$$

Similarly, the reliability of the system in Figure 3.15-b is that of a simple parallel-series system and it can be written as:

$$R(\text{system working}|\bar{B}) = 1 - [\bar{P}(A) + \bar{P}(D)][\bar{P}(C) + \bar{P}(E)]$$

Using the two previous equations, we can write the reliability of the entire system as:

$$R(S) = [1 - \bar{P}(D)\bar{P}(E)]P(B) + [1 - [\bar{P}(A) + \bar{P}(D)][\bar{P}(C) + \bar{P}(E)]]P(\bar{B})$$

3.2.3 Availability Estimation

Estimating the availability of a certain system is more involved than estimating its reliability. As we did with reliability, we need to start with few definitions.

Systems can be divided into repairable systems and nonrepairable systems. **Repairable** systems can return to an operational state after they fail by means of repair. **Nonrepairable** systems on the other hand can not return to an operational state once they fail. We will designate the *repair rate* as μ while the *failure rate* will be designated as λ.

Failures on the other hand, can be classified into detectible and non-detectible failures. **Detectable failures** can be detected and the component can be repaired or replaced. However, **undetectable failures** can not be detected. The probability of the system failing due to a detectable failure is called the **safety** of the system. In contrast, the **un-safety** of a system is the probability of failing due to an undetectable failure.

A system can be better described by its unavailability rather than its availability [42]. For example, describing a system as having 500 outage minutes per year is more appreciated than describing it as being 99.9% available.

Availability can be classified either according to the time-interval under consideration or the type of down-time (repair or maintenance). The **time-interval availability** can be

divided into: *Instantaneous (Point)*, *Average Up-Time* and *Steady-State* availabilities. The **down-time availabilities** includes: *Inherent*, *Achieved* and *Operational* availabilities.

Instantaneous (Point) Availability measures the probability that the system is operational at a random time t. **Average Up-Time Availability** measures the average probability of the system being in operation during a period $(0, T)$. **Steady-State Availability** reflects the availability of the system over a large period of time, that is $A(t = \infty)$. **Inherent Availability** includes only time to repair or replace the failed component and excludes ready time, prevention time, logistics and administrative time. **Achieved Availability** includes corrective and preventive maintenance time.

The choice of a certain availability type depends on what we want to measure. For our evaluation purposes, steady-state availability is the most feasible choice because we are interested in the long term availability of the system. For more information about other types, readers are referred to [40].

There are many approaches used to estimate availability. Examples include, Alternating Renewal Process and Markov Process Modeling. The later will be used in this study.

The Markov model is based on the idea of modeling the system as states and state transitions. At anytime, a system can be in one of two states, either faulty or fault-free.

Figure 3.16 shows a simple Markov chain for a nonrepairable system which is composed of a single component. S_1 denotes a fault-free state while S_2 denotes a faulty state. The diagram can be read as follows:

- If the system is in S_1, it will fail in Δt time with probability equals $\lambda \Delta t$.
- If the system is in S_1, it will not fail in Δt time with probability equals $1 - \lambda \Delta t$.
- If the system is in S_2, it will continue in state S_2 in Δt time with propability equals 1.

In contrast, Figure 3.17 shows a Markov chain for a repairable system. Note the appearance of a new transition with a rate (μ) that corresponds to a repair. It is important to note

that Markov principle works under the following assumptions [43]:

1. The failure rate (λ) and repair rate (μ) are constant. If the failure or repair rates are time-dependent, then the model collapses. Yet, the statement has some exceptions.

2. There can only be one transition from a state to another at any point of time.

3. The states are exhaustive which implies that at any point of time, the system has to be in one and only one state.

4. The states are mutually exclusive.

   The last two assumptions can be mathematically written as

$$P_0 + P_1 + P_2 + ... + P_n = 1$$

   where n is the number of states.

To find an expression for the availability of a particular system, we need to do the following steps:

1. Define all the mutually exclusive states of the system, e.g, a system can be functioning or faulty.

2. Build a Markov chain describing the system's states and their transitions probabilities.

3. Reduce the state diagram into a simpler one if possible.

4. Construct a transition table showing the transition rates between each pair of states.

5. Build a transition matrix using the transition table.

6. Use the transition matrix to derive the probabilistic equation of each state as follows:

$$\begin{bmatrix} P_0 \\ P_1 \\ . \\ P_n \end{bmatrix} \times \begin{bmatrix} -r_{00} & r_{01} & .. & r_{0n} \\ r_{10} & -r_{11} & .. & r_{1n} \\ .. & .. & .. & .. \\ r_{n0} & r_{n1} & .. & -r_{nn} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where r_{mn} is the transition rate from state m to state n.

7. Availability can then be calculated as the probability of the system being in fault-free states.

The previous steps are common to calculating all availabilities. The resulted equations can then be solved to get specific availability type. For example, by taking the differentials of the resulted equations and equating them to zero, we can find the steady-state availability.

Now let's see how to derive the availability equations for both non-repairable and repairable systems.

Non-Repairable Systems

We can define the probability of the system being at a certain state (S_a) as the probability that it is at (S_a) times the probability that it well not move to another state plus the probability that it is in another state (S_b) times the probability that it will move to state (S_a). Hence, we can write the probabilities for the Markov chain shown in Figure 3.16 as

$$P_1(t + \Delta t) = [1 - \lambda \Delta t] P_1(t) + 0 P_2(t)$$

$$P_2(t + \Delta t) = \lambda \Delta t P_1(t) + 1 P_2(t)$$

.

By dividing by Δt and rearranging the equations we get

$$\frac{P_1(t+\Delta t) - P_1(t)}{\Delta t} = -\lambda P_1(t)$$

$$\frac{P_2(t+\Delta t) - P_2(t)}{\Delta t} = \lambda P_1(t)$$

.

Now, by taking the limit as $\Delta t \to \infty$ we get

$$\frac{dP_1(t)}{dt} = -\lambda P_1(t) \tag{3.5}$$

$$\frac{dP_2(t)}{dt} = \lambda P_1(t). \tag{3.6}$$

The system of equations can be solved in many ways using Laplace transform, inverse Laplace transform, matrix-geometric approach, analytical perturbations method or Runge-Kutta method for solving differential equations.

Repairable Systems

Using the same definition of the probability of a system being in a certain state similar to non-repairable systems, we can write the probability of a repairable system. Now, we can express this probability for the Markov chain shown in Figure 3.17 as

$$P_1(t+\Delta t) = [1 - \lambda\Delta t]P_1(t) + \mu\Delta t P_2(t)$$

$$P_2(t+\Delta t) = \lambda\Delta t P_1(t) + [1 - \mu\Delta t]P_2(t)$$

By taking the derivative with respect to dt we get

$$\frac{dP_1(t)}{dt} = -\lambda P_1(t) + \mu P_2(t)$$

$$\frac{dP_2(t)}{dt} = \lambda P_1(t) - \mu P_2(t)$$

Here again, we end up having a system of differential equations which can be solved using on of the methods mentioned earlier.

Example of Availability Estimation

Assume we have a computer system that can be in one of three states, S_0, S_1 and S_2.

$$S_0 = \text{The computer system is working.}$$
$$S_1 = \text{The computer system has a hardware failure.}$$
$$S_2 = \text{The computer system has a software failure.}$$

$\lambda_1 = 0.0001$, $\lambda_2 = 0.0005$ and $\mu = 0.006$ correspond to the hardware failure rate, the software failure rate and to the repair rate, respectively. Let us find the steady-state availability of this system.

1. First, the description of the problem describes three states for the system: working (S_0), under a hardware failure (S_1) or under a software failure (S_2).

2. Second, we draw a Markov chain that represents the system described. This can be seen in Figure 3.18. The diagram needs no further reduction.

3. Table 3.4 shows the transition table of the system.

4. The probability equations can be derived using the following matrix operation

From | To	S_0	S_1	S_2
S_0	$\lambda_1 + \lambda_2$	λ_1	λ_2
S_1	μ	μ	0
S_2	μ	0	μ

Table 3.4: A Transition Table of the System in the Example.

$$\begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix} \times \begin{bmatrix} -(\lambda + \lambda_2) & \lambda_1 & \lambda_2 \\ \mu & -\mu & 0 \\ \mu & 0 & -\mu \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

5. We can now write the probability equations as

$$-(\lambda_1 + \lambda_2)P_0 + \mu P_1 + \mu P_2 = 0$$

$$\lambda_1 P_0 - \mu P_1 = 0$$

$$\lambda_2 P_0 - \mu P_2 = 0$$

6. To solve the above equation, we will utilize the fact that

$$P_0 + P_1 + P_2 = 1. \tag{3.7}$$

   Then, by substituting P_1 and P_2 in Equation 3.7

$$P_0 = \frac{\mu}{\lambda_1 + \lambda_2 + \mu}$$

7. Now, P_0 represents the availability of the system. By substituting the values of λ_1, λ_2 and μ, we get $A = P_0 = 0.909$. Similarly, the unavailability of the system can be calculated as $\bar{A} = 1 - P_0 = 0.091$.

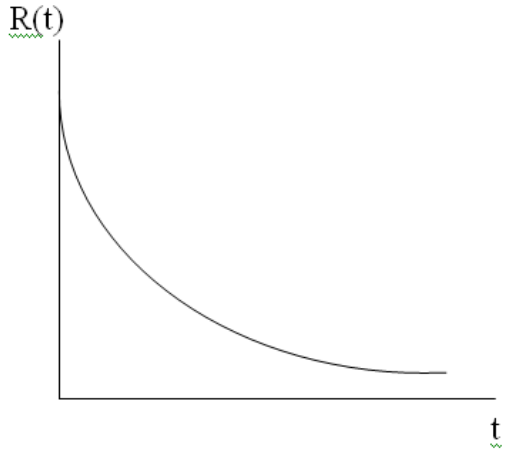


Figure 3.9: The Constant-Failure Rate Model.

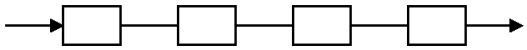


Figure 3.10: A Series System.

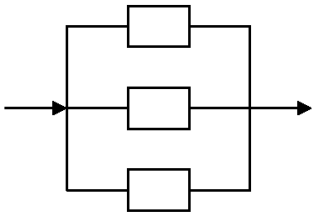


Figure 3.11: A Parallel System.

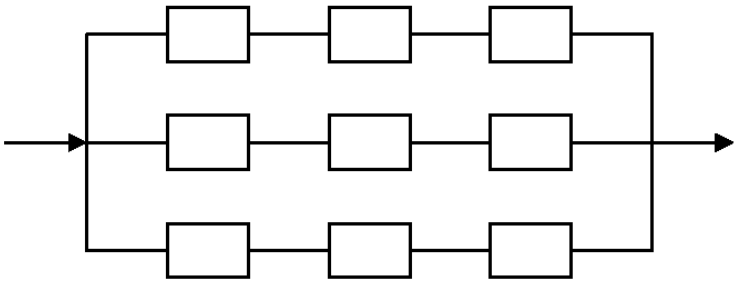


Figure 3.12: Parallel-Series System.

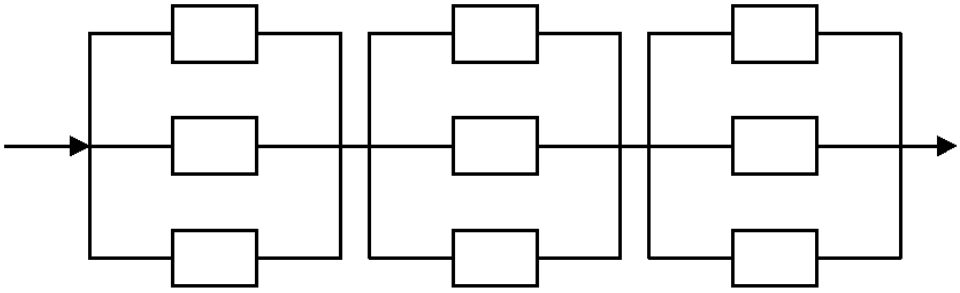


Figure 3.13: Series-Parallel System.

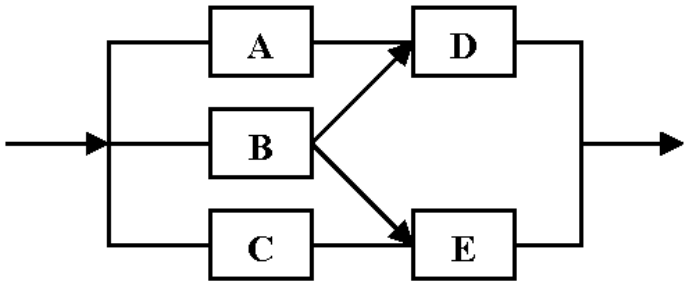


Figure 3.14: A Complex System.

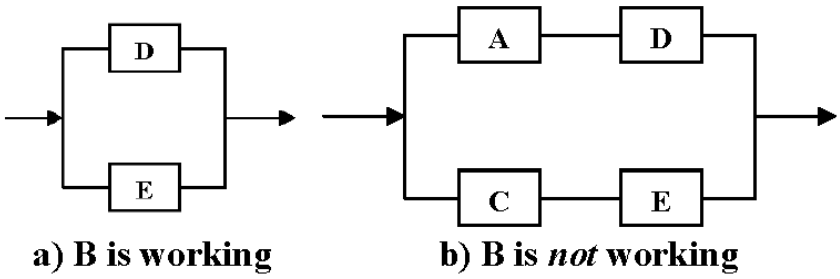


Figure 3.15: Decomposition Based on B as Module x.

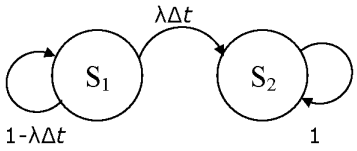


Figure 3.16: A Markov Chain for a Nonrepairable System.

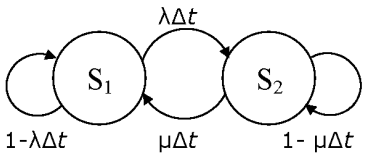


Figure 3.17: A Markov Chain for a Repairable System.

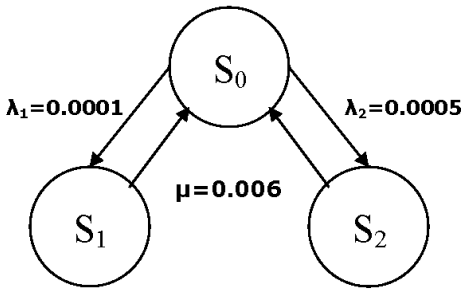


Figure 3.18: A Markov Chain for a Computer System.

Chapter 4

Related Work

Network management systems are key elements of any network. One function of network management systems is fault-management. Hence, network management systems have to be reliable and fault tolerant to be able to continue managing the network even in case of failure.

In this regard, very little work has been done in the area of network management systems' reliability. Many network management books may briefly discuss the issue without giving it the importance it deserves [44]. In this section, we review some of the related work that addresses the problem of reliability and fault-tolerance of network management systems.

4.1 A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm with Timestamps

A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm (Hi-ADSD) with Timestamps has been introduced in [45]. The Hi-ADSD with Timestamps has been used to design a distributed network monitoring tool in [4]. The tool consists of agents managing certain nodes within a network. Agents communicate using SNMP and the system can run even when there is only one fault-free agent.

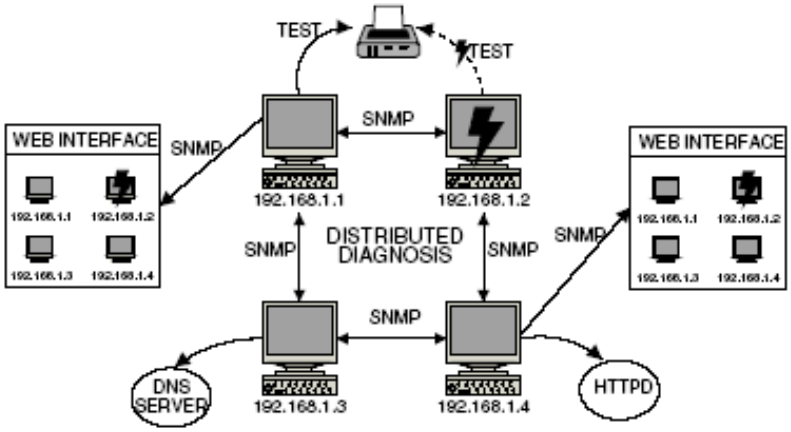


Figure 4.1: Network Management System Using HI-ADSD

Figure 4.1 shows the architecture of the system presented in [4]. The tool is composed of agents residing on network primary nodes. Each agent is responsible for a set of secondary nodes which can not run the diagnostic algorithm, such as printers and hubs. Hence, each primary node performs tests on regular intervals to check the nodes it is responsible for. In addition, each node tests the neighboring primary nodes and keeps a status of the secondary nodes tested by each neighboring node. When a neighboring primary node fails, one of the neighboring primary nodes takes the responsibilities of the failed node. This situation can be seen in Figure 4.2. In this case, node 0 takes over node 1 when the latter fails.

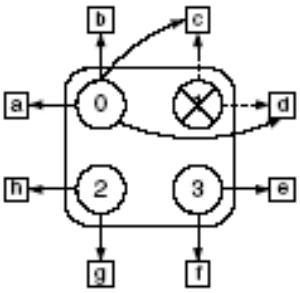


Figure 4.2: A Primary Node 0 Taking Over the Failed Neighboring Primary Node 1.

4.2 NetKeeper

An interesting approach was used in [5] to address the problem of fault-tolerance in network management systems. A tool called NetKeeper was built using *distributed object technology*. A distributed object is nothing but a natural extension of object oriented paradigm but in the context of a distributed environment.

In this technology, objects run independently constituting a distributed system. An example is a set of processes running on different machines and exchanging messages. The communication is achieved through message passing, not Remote Procedure Call (RPC) or Sockets like in traditional paradigms. The strong point in here is that to send a message a client object only specifies the server object name and the method to call. This means that the client object need not know the IP address of the machine on which the server object runs. This simplifies the fault recovery process as we will see.

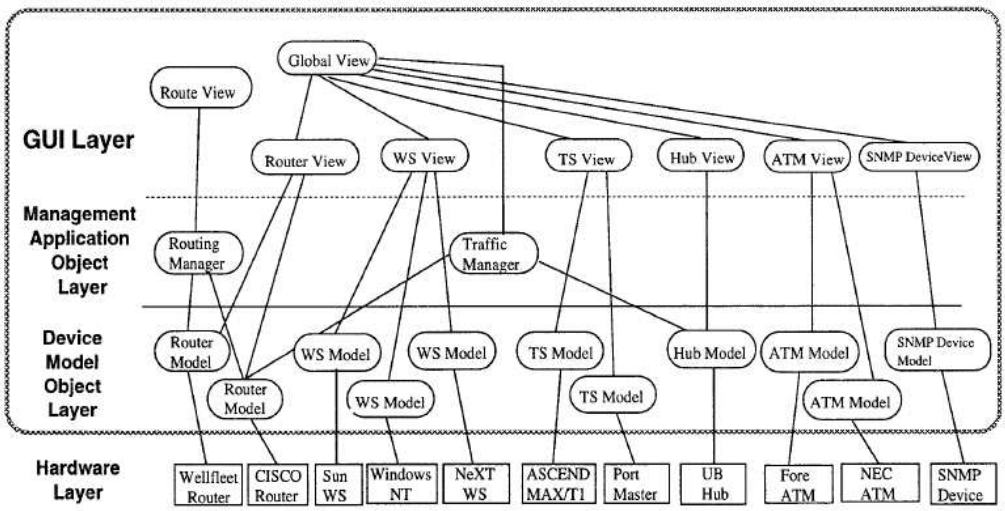


Figure 4.3: NetKeeper 4-Layers Structure

The architecture of the system is composed of four layers as shown in Figure 4.3. The lower layer, called “Hardware Layer”, models the actual physical device being managed such as a server or a router. The “Device Model Object Layer” models the agent that monitors the

hardware layer device. The device model is designed to model the actual hardware component. For example, it has the ability to poll the hardware and communicates with it using SNMP. It is also capable of telneting to the device when configuration is required.

The “Management Application Object Layer” models the manager through which the network administrator can view and interact with the device models. The ”GUI Layer” provides a user-friendly interface for administrators to interact with the NMS. Making the GUI independent from the management application makes it possible to give different views to different people based on their privileges.

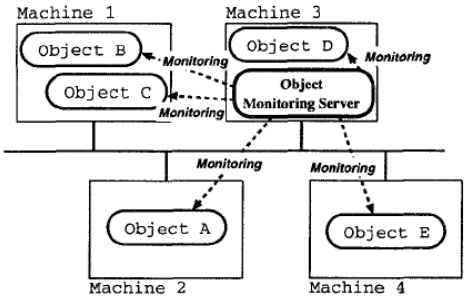


Figure 4.4: Distributed Object Environment

The system is implemented using multiple machines where each is running a set of objects. Here the word “objects” refers to management applications as well as device models. The logical environment ultimately looks as if all objects were running on a single CPU and they can communicate internally using each others’ names. An illustrative view can be seen in Figure 4.4.

Now, let us see how fault-tolerance is achieved. This is realized using *monitoring servers*. Like any other object, a monitoring server is initiated on one machine to monitor the state of other objects and take the necessary actions. When a certain machine fails, the monitoring server reruns all the processes which were running on that machine on another machine like in Figure 4.5.

In addition to the primary monitoring server, a secondary monitoring server(s) is (are)

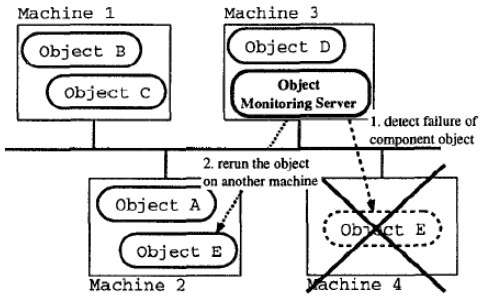


Figure 4.5: The Monitoring Server is Restoring the Management Object

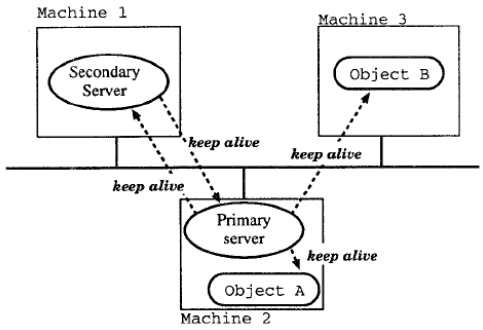


Figure 4.6: Fault-Tolerance Implemented Using Two Monitoring Servers

used to guarantee a continuous operation of the system even when the primary monitoring server fails. This is shown in Figure 4.6. The primary and the secondary monitoring servers monitor each other and if the primary fails, the secondary assumes the role of the primary server and re-initiates the primary as a secondary server on another machine. However, when the secondary monitoring server fails, the primary just reruns the secondary server on another machine as shown in Figure 4.7.

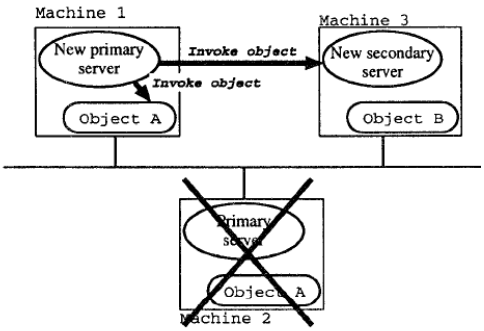


Figure 4.7: Secondary Server is Taking over the Failed Primary One

4.3 The Reliable Network Management Platform (RNMP)

Another approach for the reliability problem of network management systems has been presented in [6]. The solution presented is based on the use a 4-tier model as shown in Figure 4.8. The solution was meant to support systems supporting both SNMP and CMIP. The upper level is represented by a Management Application (MA) which plays the role of a manager sending and receiving network management data about managed objects. The MA performs fault, configuration and performance management in a GUI environment. It communicates with the next lower level using Remote Procedure Call (RPC). The MA also plays an essential role in the fault-tolerance of the network management system itself as we will see in the next few lines.

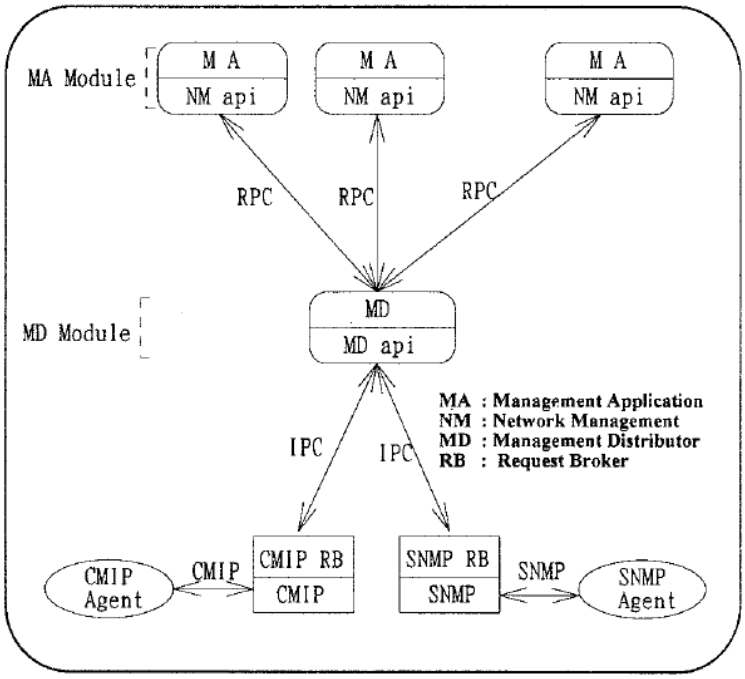


Figure 4.8: The Reliable Network Management Platform.

The next lower level is represented by the Management Distributor (MD). MDs receive requests from the MA and forward them to the appropriate Request Broker (RB) which, in

turn, forwards the request using the appropriate communication protocol understood by the target agent using Interprocess Communication (IPC). The later interaction is meant to solve the interoperability problem between different standards.

To support fault-tolerance, the MA keeps a copy of every message it sends to the MD. If the MD is down, it specifies another MD existing in the current network and sends to it the kept messages that are not yet processed. The MA checks the load of the MD before requesting its service. The MA finds the MD with the lowest workload and elects it for service.

A similar procedure happens between MDs and RBs. The MD maintains a copy of each message sent to the RB. If the RB is down, the MD finds another RB on another host and sends to it the requests that are not yet processed.

4.4 Analysis of Existing Work

The existing work presented in this chapter provides solutions to the reliability problem in NMSs but has many shortcomings. The Hi-ADSD provides a distributed solution which can manage a network even if only one primary node is fault-free. However, the solution does not specify how different primary nodes can keep track of each others' databases. Moreover, since agents run the Hi-ADSD algorithm, new agents have to be programmed.

The NetKeeper uses Distributed Objects Technology which hides from the applications many details such as the addressing scheme and the underlaying machine. The design is also modular and covers many aspects of network management. Another advantage is the easy re-invocation of failed processes on new machines. However, the work did not present how managers can coordinate and share databases. In addition, the design does not provide a centralized control of the system and uses expensive distributed object platform.

The RNMP was not meant mainly for reliability but rather for interoperability between

CMIP and SNMP. Nevertheless, some reliability aspects were considered in its design such as keeping a copy of messages sent between MA and MD or MD and RB to guarantee fault-tolerance. The later presented reliability at the middle layers only, namely MD and RB but not at the MA layer. Moreover, the new function of logging messages in the MA requires a new MA to be programmed. Furthermore, the solution neither provides a centralized control of the systems nor addresses how management information is shared or duplicated between managers. Table 4.1 presents a summary of the pros and cons of each solution.

Table 4.1: A Comparison between Hi-ADSD, NetKeeper and RNMP.

	Hi-ADSD	**NetKeeper**	**RNMP**
Paradigm	Distributed	Distributed	Hierarchical
Technology	Uses HI-ADSD Algorithm, SNMP	Distributed-Object , SNMP	New agent and RB, SNMP, CMIP
Agents	New	New	Ordinary SNMP or CMIP Agents
Manager	No (Fully Distributed)	New	New
Overhead	Inter-Agents Communication	Monitoring Servers, Hardware	MD, RP
Failure Detection	HI-ADSD Algorithm	Distributed-Object Tool	No detection
Database Synchronization	Not Mentioned	Not Mentioned	No need
Agent-Agent Agent-Manager Communication	SNMP	Message Passing, RPC, Socket	RPC, IPC
Advantages	Can run even if only one agent is fault-free	Hides details from application & Customizable	Works with both SNMP & CMIP
Disadvantages	Requires new agent and no database replication	No centralized view/control, no database replication new manager needed and expensive	No reliability at MA level, no database replication and requires new MA

Chapter 5

The Proposed Framework

As stated in the introduction, the objective is to design a fault-tolerant Network Management System. In addition to being fault-tolerant, it is also required to have an additional set of features to be more competitive.

The proposed system is called Fault-Tolerant Network Management System (FTNMS) which is shown in Figure 5.1. It consists of two parts. Each part deals with the reliability of certain level in the hierarchy of the network management system. As we have seen in Chapter 2, a general network management system may consist mainly of two or three layers:

- A Manager-of-Managers (MoM),
- Mid-Level Managers (MLM), and
- Agents.

The first part of the proposed system addresses the reliability of the MoM while the second deals with the reliability of the MLMs. It is to be noted that the first part can be applied to centralized and hierarchical NMSs while the second can be applied to hierarchical and may be extended to distributed NMSs. In the rest of the chapter, part 1 will be called FTNMS-MoM and part 2 will be called FTNMS-MLM.

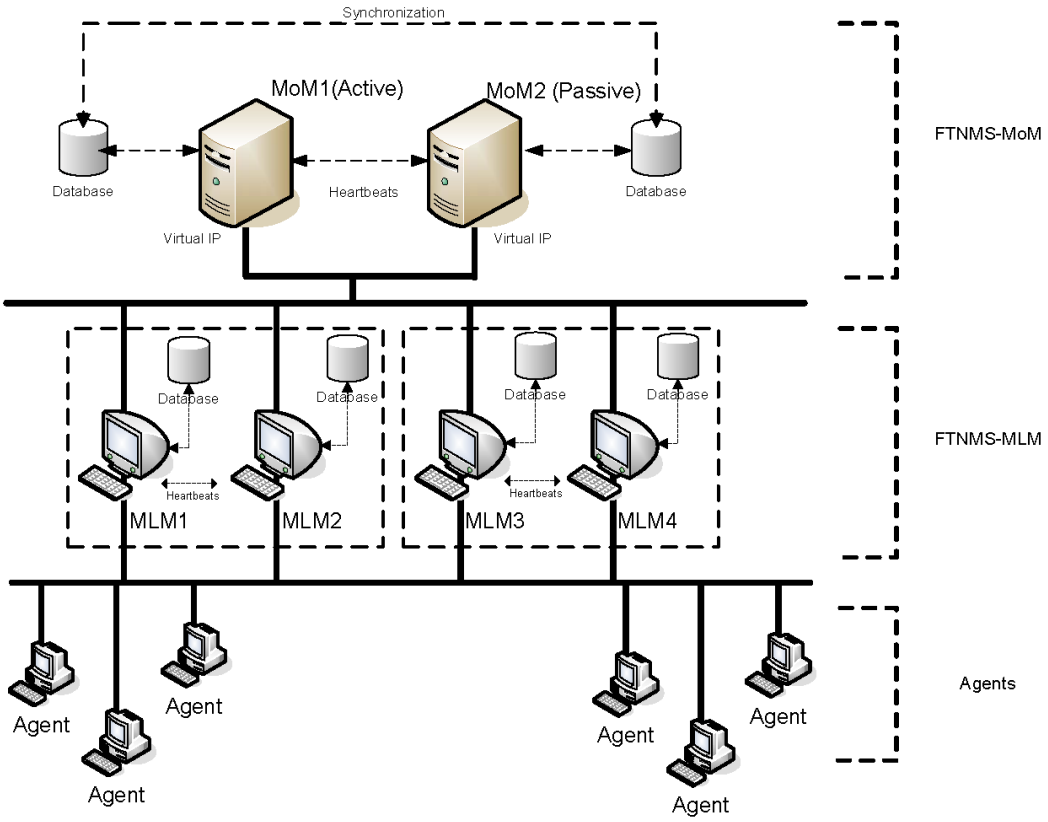


Figure 5.1: FTNMS: A Hierarchical View.

In the following section, we will discuss the system requirements, the design methodology, design details and issues.

5.1 Requirements

The system was designed with the following requirements in mind:

1. It must be fault-tolerant in the sense that it can continue working even under the failure of some of its components.

2. It must be viewable to the administrator as one system, i.e., the administrator must have a centralized view of the whole system.

3. It must provide a unified system image, meaning that it integrates all the distributed databases into one that represents the whole system.

4. It must not only address centralized network management, but extends to hierarchical and distributed network management as well.

5. It must be scalable, meaning that it can be scaled up or down depending on the organization's needs.

6. It should require as fewer changes as possible to the existing protocols and hence can be incorporated seamlessly into existing systems.

7. It must use standard tools and protocols.

8. It must be affordable to suit small organizations.

Although some of the requirements mentioned are conflicting, the proposed system could satisfy them all as we will see in this thesis.

5.2 Design Methodology & Decisions

Let's take a look on how the above requirements have been met:

- The fault-tolerance of the MoM has been achieved by using two MoMs, with one acting as a full backup with a full and up-to-date copy of the database. The backup keeps listening for heartbeats from the active manager and synchronizes its database with that of the active. In the MLM layer however, each two MLMs are grouped into a pair and each paired MLMs back up for each other.

- A unified view of the whole system has been achieved by using the active/passive mode on the MoM level with the active database being on the active MoM. This allows the administrator to have a centralized view and control on one machine.

- Single addressing of the MoM has been achieved by using an active/passive configuration with a single Virtual IP (VIP) address that is assigned to the active MoM. This also allows for a transparent incorporation of the system into any existing NMS application without changing the protocols or the code of the application. In addition, if for some reason, one decides to implement only the MoM part of the framework, the agents with which the NMS is communicating will not be aware of the change. Furthermore, allowing the MLMs to acquire the IP of their partner upon failures hides these failures from agents. The addressing schemes used in FTNMS-MoM and FTNMS-MLM allow the design to fit with existing protocols without the need for any modifications.

- Using MLMs allows for the framework to be extended to hierarchical network management and may be used to enable distributed network management. By placing nodes in different parts of the network, it is possible to manage large networks with FTNMS.

- The scalability plus the performance concerns of the framework have been remedied by implementing the MLM which relieves the MoM from dealing with individual nodes and hence maximizing the scalability. Moreover, MLMs can be delegated some tasks and this increases the cumulative performance of the system.

- The transparency of the MLM has been achieved by allowing the backup MLM to acquire the IP of the failed MLM and hence, neither the MoM nor the agent will be affected by the failure.

- Compared to letting all MLMs back up for each other, grouping MLMs into pairs minimizes the bandwidth used by the heartbeat and database synchronization modules without losing the functionality meant for achieving HA.

- All the communication between the MoMs, MLMs and agents has been left to SNMP or whatever protocol used. This independency allows the solution to be migrated with

any standard tool without requiring any change.

- The cost of the solution, software-wise, is totaly free. All the tools used are freely available and widely supported. In addition, all are open source software, which added portability and extendibility to the framework. It is also interesting to note that for organizations with already existing hierarchical NMS, it is possible to turn their system into a reliable system by adding only one MoM node and grouping MLMs. This implies roughly the cost of one machine.

As a bi-product, the system can be used with a custom centralized NMS application and agents without requiring a new software to be developed. Yet, the MLM can be customized if needed. Hence, even if one has a closed-source NMS, he can still extend it using the proposed framework by modifying the MLM code.

Certainly, the advantages gained do not come at no cost. The costs will come mainly from the extra hardware needed for the back up MoM besides the extra traffic generated from exchanging the heartbeats and synchronizing the databases.

Table 5.1 summarizes how each objective was met in the proposed design.

Table 5.1: How FTNMS Achieved Different Design Objectives.

Objective	**Approach**
Centralization	Using MoMs in Active-Passive mode
Reliability	Using backup MoM and pairs of MLMs
Seamlessness	Using VIP and MLM layer
Affordability	Using free open source software *heartbeat* & *DRBD*
Scalability	Using MLMs

The reasons behind using this architecture is its simplicity and ability to fit with the topology of today's hierarchical networks. Actually, during the design, another alternative was considered which is using a fault-tolerant database server to which all MLMs report instead of each MLM maintaining its own database. This approach was rejected because it will increase the traffic enormously in the network specially on the segment where the database

server resides since each MLM will be receiving data and will also send it to the database server. In addition, if the database server is isolated because of a failure in the network, the manager will not be able to retrieve information. Furthermore, the database server will need a mean to combine different pieces of information received from different managers.

5.3 Part I: FTNMS-MoM

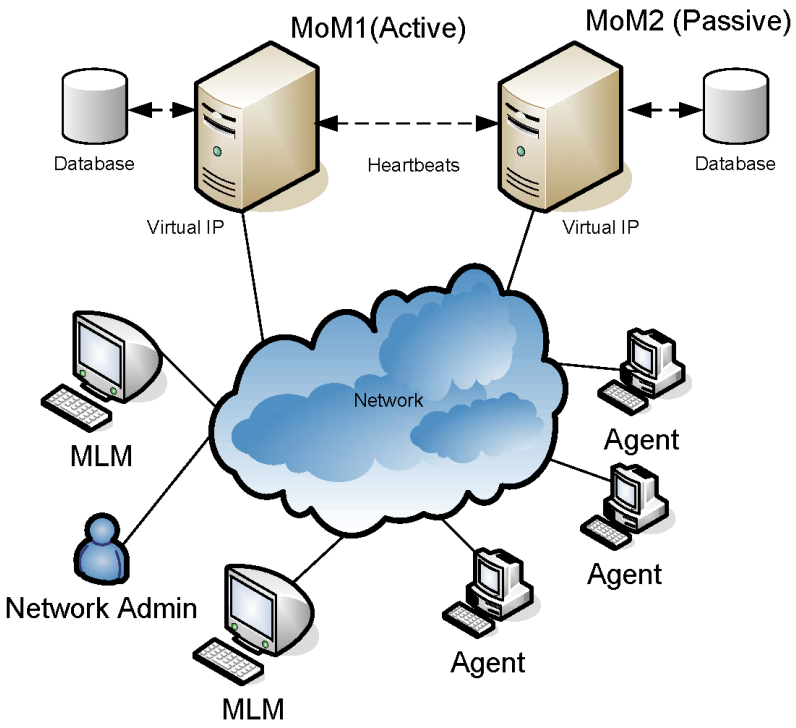


Figure 5.2: FTNMS: Part 1: FTNMS-MoM .

Figure 5.2 presents FTNMS-MoM. The system consists of two managers, an active MoM and a passive (standby) MoM. The two MoMs constantly exchange heartbeats through an Ethernet channel to monitor the status of each other. In addition, the two MoMs maintain two synchronized databases. The databases exist physically on both MoMs and are updated frequently. Only the active MoM is allowed to update the database, the passive MoM keeps

receiving all updates and incorporating them to its own copy.

The two MoMs have two real IP addresses, RIP1 and RIP2. In addition, the two MoMs have a common Virtual IP (VIP) that is assigned only to the active MoM. The VIP is a floating IP, meaning it can be moved from one MoM to the other. The VIP is the IP that is used to address the MoM by other entities such as MLMs, agents or network administrators. But, if the network administrator is to access the passive MoM, he or she has to use the RIP of that MoM.

The idea behind using an active/passive mode rather than an active/active one is due to two reasons:

1. Allowing only one node to be active permits us to have a centralized addressing scheme, i.e., regardless of which MoM is active, administrators, MLMs and even agents can reach the MoM using one single IP. But, if we are to use two active MoMs, we will lose centralization.

2. Getting a unified data view. Since all the data is in one place, administrators can see a unified picture of the whole system, rather than a partial picture on each MoM.

On the other hand, having an active/passive mode will certainly decrease the utilization, since the passive node is sitting idle.

When the active manager fails, the passive manager becomes active and it claims the VIP. This process includes assigning the VIP to an alias interface and sending gratuitous ARP replies to other nodes to change the mapping of the VIP to its own MAC instead of the MAC of the failed manager. Since, the passive manager has an up-to-date copy of the database, it can continue serving instead of the failed manager.

Using the same IP address hides the failure from the lower level entities (MLM, agents and network administrators) as opposed to informing all the involved entities about the new manager.

5.4 Part II: FTNMS-MLM

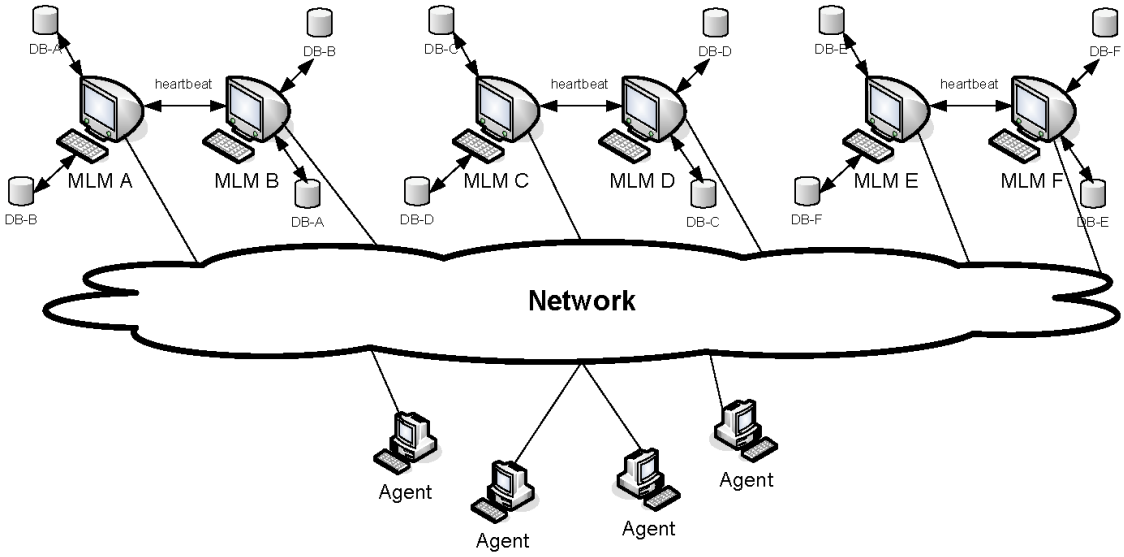


Figure 5.3: FTNMS: Part 2: A Logical View of FTNMS-MLM .

Figure 5.3 presents the FTNMS-MLM which consists of pairs of MLMs. Each pair consists of two MLMs that are backing up each other. This means that if one MLM fails, its partner will be responsible for managing its agents. To enable this, each manager maintains two databases, one of its own and another one representing a copy of the database of its partner. Each MLM has also to know the IP of its partner. In contrast to the MoMs, each MLM has a different IP address from than its partner.

When an MLM fails, its partner will assign the IP of the failed MLM to its interface and continue monitoring the agents for which the failed MLM was responsible. All information gathered from those agents will be appended to the database of the failed MLM. The agents will be unaware of the changes happening since they are still using the IP address of the failed MLM.

When the failed MLM is up again, the partner MLM will release the IP address and the

database. The database will be also copied from the partner to the recovering MLM. The MLM can now continue working as if nothing had happened.

MLMs can be delegated some tasks by the MoM. For example, the MoM can request an MLM to report the status of some nodes in the network. By adapting the failover scheme as we have just described, the MoM will not feel the difference when a certain MLM fails because it can still use its IP address and get results back.

Choosing the MLMs to be working in an active/active mode serves in aiding the hierarchical nature and consequently the scalability of the framework. Moreover, having the MLM in pairs simplifies the framework and lowers the bandwidth and space needed to exchange the heartbeats, and to synchronize and store the databases. Using the floating IP hides faults from the agents and the MoM.

The MLMs are designed with the following features:

1. Each MLM has an IP address that is different from, but known to, its pairing MLM.

2. Each MLM maintains a copy of its database in addition to the database of its pairing MLM.

3. Each MLM logs each operation before it starts it, and indicates when it is successfully completed in its database. This allows the taking over MLM to resume any uncompleted action.

4. For the scope of this work, partitioning nodes among MLMs will be considered static. The main goal of this work is to show that the framework works.

5. During the failover process of the MoM, MLMs will have no mean to reach the MoM. In ordinary NMSs, the MLM operation will timeout and the information will be lost. In FTNMS, MLMs will try to resend the data they have until the failover is complete allowing for no loss of data even during failover.

Chapter 6

System Implementation

The following section will present the implementation details of the FTNMS. To understand this part, it is assumed that the material covered in Chapter 2 is understood.

6.1 Implementation Decisions

The implementation platform was chosen to be Linux because of its: open-source nature, flexibility, and robustness. Linux is also supported by many tools for similar researches. In addition, Linux is preferred for servers and NMS because it is less affected by security issues.

The High-Availability tool was selected to be the Linux Heartbeat. The decision was made due to the maturity of the tool, ease of interfacing, and open-source nature. The database synchronization tool was chosen to be DRBD since it is also open-source and was meant to work with Heartbeat. The selection of these tools came also from the fact that the design must be affordable. It must not be thought that using free software compromises the efficiency and the robustness of the design. The tools were selected mainly because of their high maturity and dependability. Hence, many of them are used in commercial products [23].

The implementation involves installing and configuring a set of virtual machines using UML for testing. More details about the installation and configuration can be found in Ap-

pendix D. Then, each machine was configured based on its role in the system. For instance, two machines were configured as MoMs. For experimentation, only two machines were configured as MLMs. Other machines were configured as managed objects, hence SNMP agents were installed on them. On both MoMs and MLMs, we have Heartbeat and DRBD installed.

Using UML here is for testing purpose only. A real system is implemented using real hardware for sure due to performance considerations. UML has been chosen for the following reasons:

1. The cost incurred in the needed hardware was not clear at the beginning of the study. Using UML will also remove any limit on the number of machines used.

2. Dealing with advance Linux features such as kernel patching and kernel recompilation, disk partitioning, etc. would be a risky process and might require the installation of the whole system again each time it is being damaged.

3. Simplicity in acquiring any extra hardware. For example, all what we need to add a new disk partition to UML is to create a file system file and attach it to UML.

Actually, after testing the active-passive MoM system on UML, it was deployed on a real physical system and the testing was successful.

Now, let us discuss two different parts of the implementation:

1. FTNMS-MoM Implementation

2. FTNMS-MLM Implementation

More details about each implementation are presented in the following sections.

6.2 FTNMS-MoM Implementation

The MoMs were implemented using two UML instances. Both instances have a daemon of the heartbeat application which continuously exchange heartbeats. The active MoM, uses a

DRBD disk in a “Primary” mode as a database storage. The passive MLM however uses a DRBD disk in a “Secondary” mode. A web-based NMS application was installed on both MoMs and can be accessed by pointing the web browser to the VIP.

When the working MoM fails, the standby MoM will become active and take over all the resources of the failed NMS, including the IP address, the NMS application, the shared disk controller (DRBD) and the web server.

The DRBD allows only the MoM in “Primary” mode to take control of the shared disk, with read and write privileges. For any write operation on the shared disk in the active MoM side, a similar operation will be carried out on the passive MoM side to synchronize the two disks. While the active MoM is alive, the applications on the passive MoM can not mount the shared disk, hence they can neither read nor write to the shared disk. Synchronizing the shared disk is the responsibility of the DRBD daemon on the passive MoM.

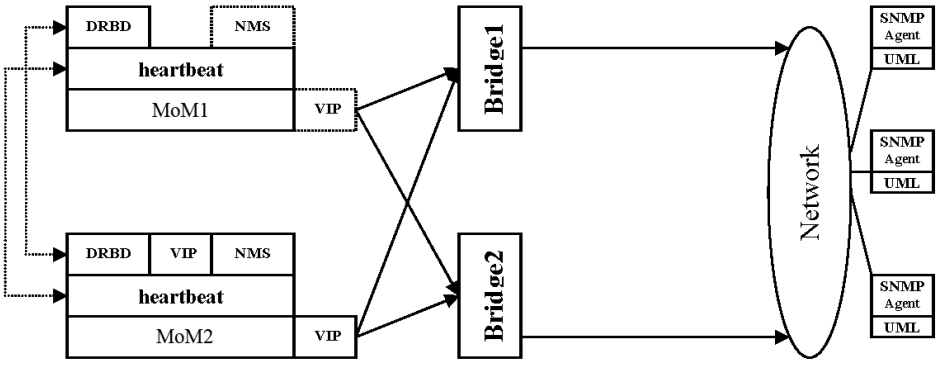


Figure 6.1: The Implementation of FTNMS-MoM in UML.

A logical diagram of the implementation is shown in Figure 6.1. When the passive MoM stops hearing heartbeats from the active MoM, it does the following steps:

1. Create an alias of its working Ethernet interface , e.g., (eth0), and assigns the VIP to it.
2. Sends gratuitous ARP replies to change the IP-MAC mapping on all nodes to map to its MAC rather than the failed MoM’s.

3. Mounts the shared disk, and assigns itself as the “Primary” node.

4. Runs the web server and NMS application.

The implementation is simple and satisfies the requirement of many organizations. We have chosen a simple and free web-based NMS to test the setup. The NMS is called nPulse and can be found in [46]. The added advantage of having a web-based NMS is to be able to access it from anywhere.

Exact details about how each tool is installed and configured can be found in Appendix-E. Few snapshots of the nPulse interface can also be seen in Appendix-B.

6.3 FTNMS-MLM Implementation

The implementation of the MLM is more involved than the implementation of the MoM since it involves writing the code of the MLM as well as installing and configuring the heartbeat and the DRBD tools. In this section, we will discuss the details of the MLM design and the configuration of the HA tools. The configuration for the HA tools is different from that of the MoM since MLMs are working in an active/active mode.

To allow for an easy adminstration of the MLM, a special applet was written through which the administrator can view the status of and configure the MLM. A snapshot of the applet interface can be seen in Appendix-B.

The MLM application was written using Java. The choice was driven by two factors. The first wa portability and the second was the availability of many Java libraries for SNMP implementations. This work benefited from two JDKs: SNMP Java Package[47] and Mibble[48].

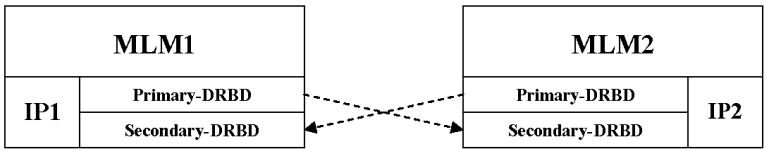
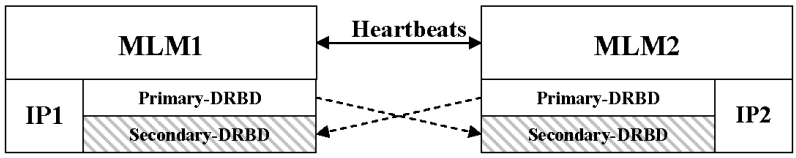


Figure 6.2: The Implementation of One FTNMS-MLM Pair in UML.

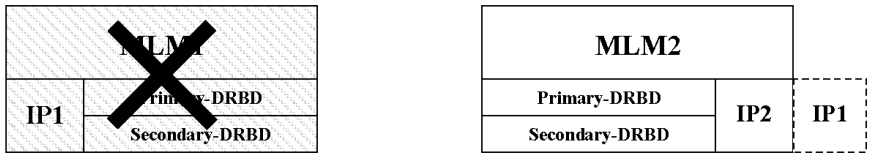
6.3.1 HA Tools Configuration

The configurations of the HA tools in the middle layer, that is the MLM layer, is based on the active/active mode. Figure 6.2 shows how a pair of MLMs would be implemented. Each MLM uses a different IP, IP1 for MLM1 and IP2 for MLM2. In addition, each MLM maintains two DRBD disks, a Primary-DRBD associated with it, and a Secondary-DRBD associated with the partner MLM. As in Figure 6.3-a, Each MLM works on the Primary-DRBD only, while the Secondary-DRBD is synchronized with the Primary-DRBD of the partner MLM. In addition, Each MLM has only one IP address associated with it which is different from its partner's IP address.

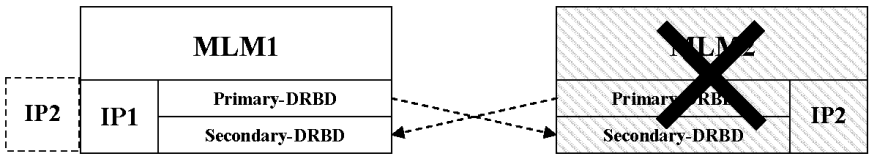
When MLM1 fails, MLM2 detects that through the heartbeat daemon. Then, MLM2 assigns IP1 to its interface. Since MLM2 has a copy of the DRBD disk of MLM1, it activates the Secondary-DRBD disk and starts using it as if it was MLM1. A similar operation happens in MLM1 when MLM2 fails, see Figure 6.3-c.



a) Both working: Only Primary-DRBD is Active on Both MLMs.



b) MLM1 Fails: MLM2 Acquires Secondary-DRBD and IP1.



c) MLM2 Fails: MLM1 Acquires Secondary-DRBD and IP2.

Figure 6.3: The Failover Operation of the MLM.

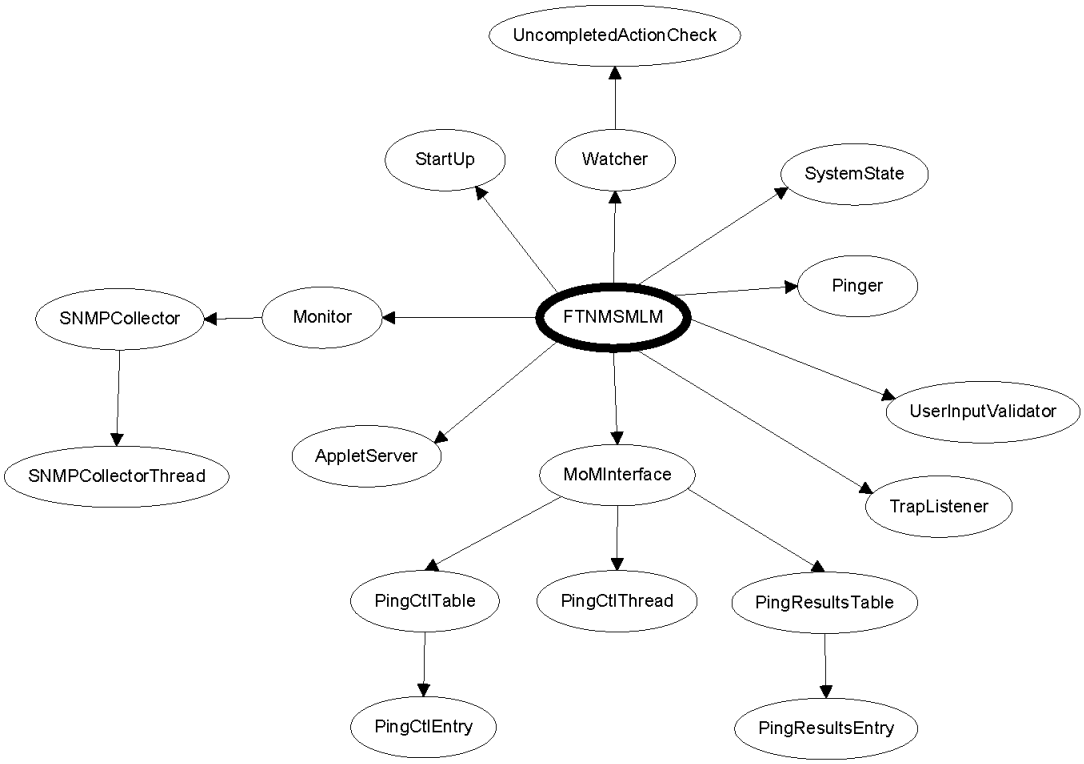


Figure 6.4: FTNMS-MLM Classes.

6.3.2 MLM Programming

The FTNMS-MLM has been implemented using many classes. Some classes have also been made as threads to allow functions to be running concurrently. Figure 6.4 shows the interaction and the invocation sequence of classes.

Before we start the description, let us list the capabilities of the MLM:

1. Each MLM keeps monitoring the agents and maintains a database about their status.
2. Each MLM listens to traps received from agents and forwards them to the MoM.
3. Each MLM can be accessed and controlled via a web browser even when it is down.
4. Each MLM monitors the status of its partner and takes over when the partner failed.

5. Each MLM listens to requests from the manager to do remote operation requests. When finished, results are sent back to the MoM.

6. Each MLM has the capability of reading logs of the failed partner and resuming any uncompleted action.

Next, we will discuss the main classes and the way they work. The main implementation points and techniques will also be presented. Table 6.1 lists the configuration files used by the MLMs.

Table 6.1: List of Configuration Files Used with MLM

File	Details
ip.mlm	The IPs of the MLM and its partner
mom.mlm	The IP and the community string of the MoM
drbd/stat.mlm	Collected statistics
drbd/agents	Agents to monitor
drbd/tmp/	Contains temporary logs of requests
drbd/traps.mlm	Logs the traps received
drbd/values.mlm	The OID of the objects to be monitored

FTNMSMLM Class

`FTNMSMLM` is the main class of the MLM application. It instantiates all the instances of the other classes. The class maintains many important objects for the operation of the program such as the **agentsList** which contains the list of agents to be monitored and the statistics collected from them. Another important parameter is **mode** which represents the mode of operation which can be *Single* when only one MLM is working and *Dual* when both MLMs are working.

The operation of the `FTNMSMLM` is summarized as follows:

1. When it first starts, it instantiates an instance of the `StartUp` class and waits for it to check for the resources (IP address and DRBD disk) to be available. If the `StartUp`

instance terminates, the program continues. Otherwise, the program notifies the user about the unavailability of resources and terminates.

2. If the resources are available, the `FTNMSMLM` reads the list of agents to be monitored from the file *agents.mlm*, and its IP and the IP of the partner MLM from the *ips.mlm* file.

3. The `FTNMSMLM` instantiates instances of the following classes

   - `Monitor`
   - `Watcher`
   - `MoMInterface`
   - `Pinger`
   - `TrapListener`
   - `AppletServer`

In addition, the `FTNMSMLM` handles user requests to start/stop monitoring and view the status of the MLM. Details about the classes can be found in Table 6.2.

StartUP Class

The `StartUp` class allows the MLM to check the status of the HA tool before the MLM starts. It checks the output of the HA tool to determine the status of the MLM. If the IP and the DRBD disk are acquired, its terminates correctly. Otherwise, it waits for 1 minute and terminates with error.

Watcher Class

The `Watcher` class watches the status of the HA tools for any change. The changes can happen whenever the partner fails or comes up after a failure. This is simply done by checking

Table 6.2: MLM Java Classes.

Class	Details
FTNMSMLM	Main class representing the MLM
MoMInterface	Interface to MoM, processing MoM's requests
StartUp	Checks for resources at startup (IP and DRBD disk)
UncompletedActionCheck	Checks failed MLM log for uncompleted requests
Monitor	Polls SNMP objects and collect statistics
TrapListener	Receives Traps from agents and forwards them to MoM
Pinger	Pings a node and report results
Watcher	Monitors the status of the other MLM and switches between single/dual modes
SNMPCollector	A thread to collect SNMP values for all managed objects
SNMPCollectorThread	A thread to collect SNMP values form one node
UserInputValidator	Checks IPs validity
PingCtlTable	Represents a PingCtlTable MIB table
PingCtlEntry	Represents a PingCtlTable MIB table entry
pingCtlThread	A thread for a PingCtlTable request
PingResultsTable	Represents a PingResultsTable MIB table
PingResultsEntry	Represents a PingResultsTable MIB table Entry
SystemState	Stores MLM mode of operation

the output of the HA tools every 30 seconds for any change.

If a failure is detected, the class reads the new agents from the image corresponding to the partner MLM (secondary-DRBD). It also instantiates an instance of an `UncompletedActionCheck` class to read the **temp** directory of the failed partner for any uncompleted requests. If any uncompleted action is found, a `PingCtlThread` is instantiated for it. On the other hand, if the partner recovers, the class removes those agents from the agents list.

Pinger Class

The `Pinger` class contains methods for pinging nodes and returning the status back. It is used when first acquiring the list of agents to add their status in the **agentsList** table.

Monitor Class

The `Monitor` class monitors the agents specified in the **agentsList** object. The instance starts by reading the OIDs from the *values.mlm* file. Then, it instantiates a thread of the `SNMPCollector` class. The SNMPCollector thread creates a thread of the type `SNMPColl-`

`ectorThread` to contact the agents, retrieve the values specified, and store them in the agentList. The values received are also logged in the *stat.mlm* file.

Monitoring can be stopped and started by suspending and running the Monitor object.

MoMInterface Class

The *MoMInterface* class is responsible for the interaction with the MoM. The class Listens to requests from the MoM and responds to them.

When the *MoMInterface* receives requests to set certain MIB values in the *PingCtlTable*, it handles those requests. If a ping request is activated, the `MoMInterface` instantiates a thread of `PingCtlTable` type to do the request, updates the corresponding row in the *PingResultsTable* and sends a notification to the MoM to inform it of the completion of the request.

TrapListener Class

The `TrapListener` keeps listening for traps on port 162. Whenever a trap is received, it gets logged in the *traps.mlm* file of the corresponding MLM to which the agents sending the trap is associated. The trap is also relayed to the MoM. If the trap is relayed while the MoM is in transition, the trap is resent after sometime. The later accounts for the case when a transition is taking place between the two MoMs due to a failure. Therefore, the trap is delayed till the transition is complete.

AppletServer Class

The `AppletServer` class is responsible for the interaction with the applet to control the MLM. Depending on the command received from the applet, the `AppletServer` stops, starts the monitor or reports the status of the agents and the MLM to the applet.

Chapter 7

Experimentation & Testing

The framework was tested under the setup shown in Figure 7.1. Table 7.1 summaries the status and configurations of each node in the setup. In this section, we will demonstrate few test cases that show the operation and reaction of the system under different situations.

7.1 Normal Operation

As in Table 7.1, during normal operation MoM1 is active while MoM2 is passive. The VIP is assigned to MoM1 and hence whenever a request from an administrator or an MLM goes to the address 172.16.70.150, it will be received by MoM1. The ARP mapping of the VIP is made with the MAC address of MoM1. Figure 7.2 shows a pictorial representation of that situation.

Table 7.1: Nodes & Their IP Addresses and Role.

Node	Status	RIP	VIP		DRBD	
			VIP	Status	DRBD	Status
MoM1	Active	172.16.70.151	172.16.70.150	Assigned	drbd0	Primary
MoM2	Passive	172.16.70.152	172.16.70.150	Not assigned	drbd0	Secondary
MLM1	Active	172.16.70.171	No	No	drbd0 drbd1	Primary Secondary
MLM2	Active	172.16.70.172	No	No	drbd1 drbd0	Primary Secondary

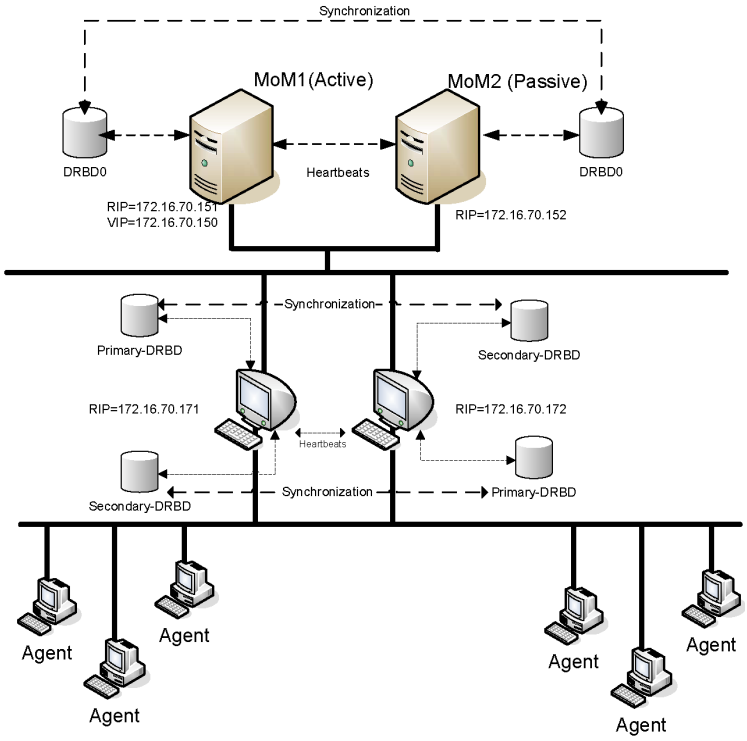


Figure 7.1: FTNMS: Testing Setup Under UML.

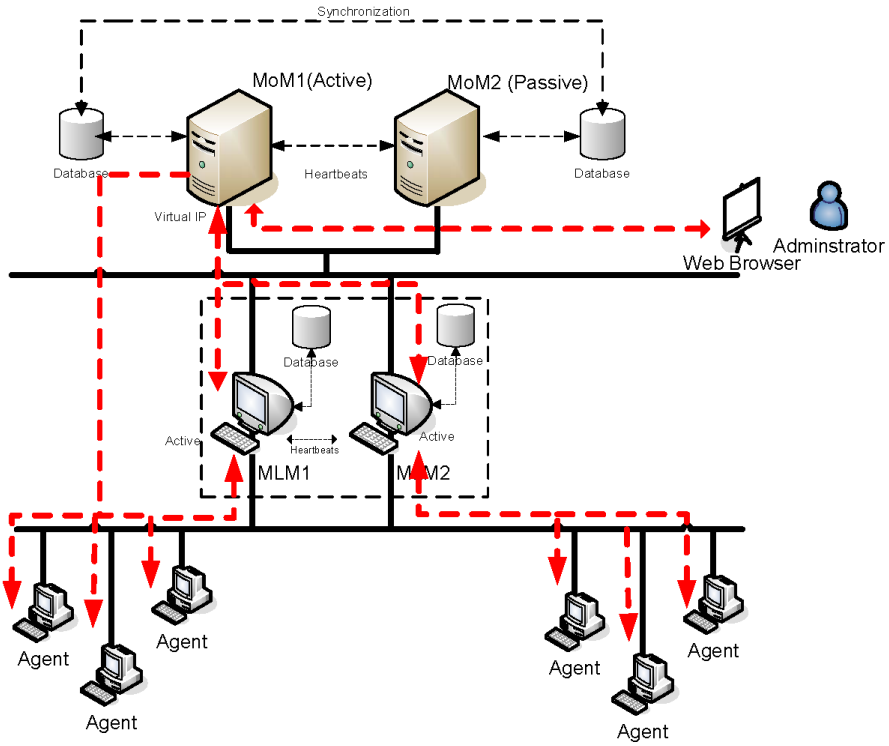


Figure 7.2: FTNMS: Normal Operation of MoM.

Similarly, every agent will contact its designated MLM using its IP address. Again, the IP address of each MLM is mapped to its MAC address. Figure 7.2 shows how the interaction happens during normal operation.

7.2 MoM Failure

There could be more than one scenario for an MoM failure:

1. When MoM1 fails, MoM2 will detect that after certain time period. The delay before detecting the failure will depend on the configurations of the HA tool. As we have seen in Section 3.1, the *heartbeat* application defines a *deadtime*, which effects the failover time. We will discuss the delay in the next chapter. When MoM2 is up, it will claim the VIP address and resume the monitoring on behalf of MoM1. See Figure 7.3.

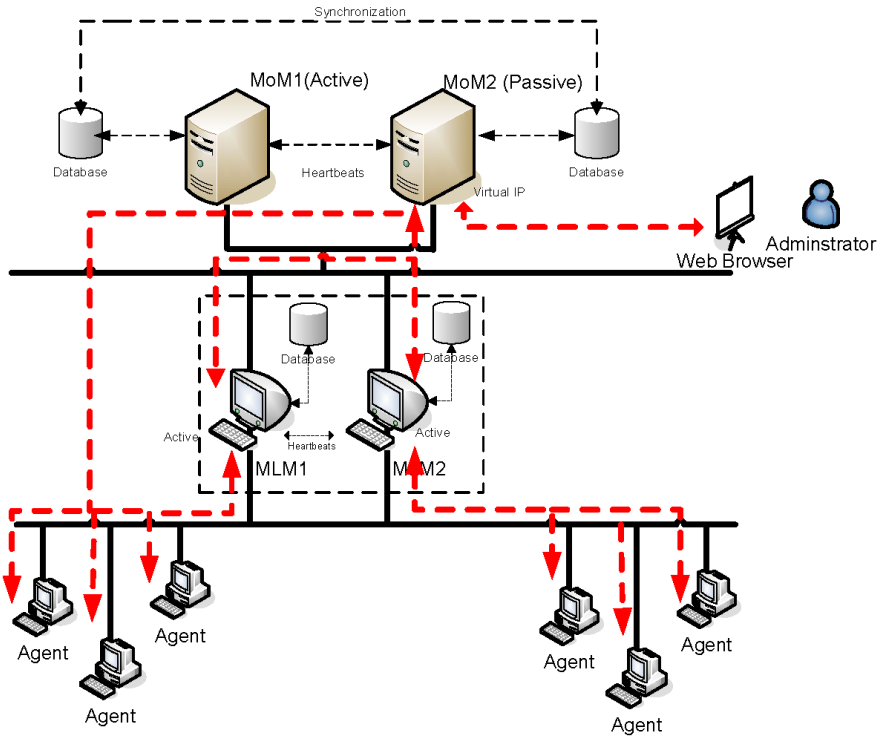


Figure 7.3: FTNMS: MoM1 is Down, MoM2 is Up.

2. If MoM1 has a partial failure that causes the watchdog to reboot it, then, MoM2 will take the lead similar to the previous case, but:

   - If the auto_failback is enabled, MoM1 will take over MoM2 after a successful reboot.
   - If the auto_failback is disabled, MoM2 will continue as active and MoM1 will become passive after successfully rebooting.

3. If MoM2 fails while MoM1 is active, the heartbeat will do no action. Moreover:

   - If the failure is a partial failure causing MoM2 to malfunction, the watchdog will detect it and reboot the machine.
   - If the failure is a complete failure causing MoM2 to shutdown, the NMS appli-

cation on MoM1 will detect the failure since MoM2 is monitored like any other node in the network using SNMP agent in addition to being a manager using an SNMP agent.

It is better not to enable the auto_failback feature unless we have a strong reason to do so because it will cause another transition interval causing a loss of active connections.

Table 7.2 summarizes all the possible scenarios and the way the system will behave if they occur.

Table 7.2: Failure Scenarios in the MoM Level.

Failing Node	**Failure Type**	**auto_failback**	
		on	**off**
MoM1	**Partial**	*Softdog* will detect the failure and reboot, MoM2 will become active. When MoM1 is up again it becomes active	*Softdog* will detect the failure and reboot, MoM2 will become active. When MoM1 is up again it becomes passive
	Full	MoM2 will become active. When MoM1 is up again it becomes active	MoM2 will become active. When MoM1 is up again it becomes passive
MoM2	**Partial**	*Softdog* will detect and reboot, NMS on MoM1 will report failure to admin	
	Full		

The case of the two MoMs failing is not addressed due to two reasons:

1. The reliability of the system is assumed to be bounded by the reliability of the network. Hence, the system will not increase the reliability of the network. That is, such a case will happen most probably when the whole network goes down, and here we are limited by the network resilience.

2. Also, it depends on the placement of the two MoMs. In other words, the placement shown in Figure 7.1 is logical and can be physically implemented in different ways by putting each MoM in one side of the network and hence lowering the probability of both of them failing.

Let's address what happens if the failure occurs while a transaction is taking place. The behavior will be mainly affected by the protocol used in the transaction:

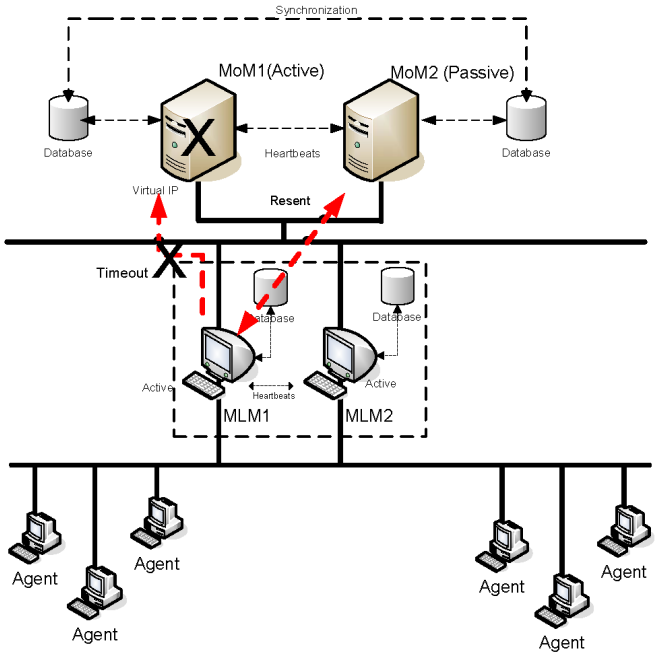


Figure 7.4: FTNMS: An MLM Reports to MoM1 While it is Down.

1. **SNMP Transactions** Since SNMP utilizes the UDP connectionless protocol, the communication is not reliable and messages may be lost. However, the SNMP application layer protocol can handle errors and timeouts but it is implementation specific. If the transaction involves regular polling of information from certain node, it will not be significant since it will be taken care of by the next poll. On the other hand, if the transaction involves receiving a notification from an MLM through a trap for instance, and a response is expected from the MoM, then the trap communication will timeout. Note, that in some implementations *Inform-Request* is used instead of *Trap* when reliability is of concern. Fortunately, in the MLM implementation, all traps are implemented as *Inform-Requests* and logged before they are sent and hence it can be checked later for any lost notifications. More importantly, the MLM retries sending the traps after timeouts to handle timeouts that happen because of a failover delay, see Figure 7.4.

2. **HTTP Transactions** Since HTTP uses TCP which is connection-oriented and HTTP is

a stateless protocol, if the manager fails while the administrator is sending requests, the request will timeout. During the failover period, the administrator might have no connection, but once the backup MoM is up, the information can be sent again.

7.3 MLM Failure

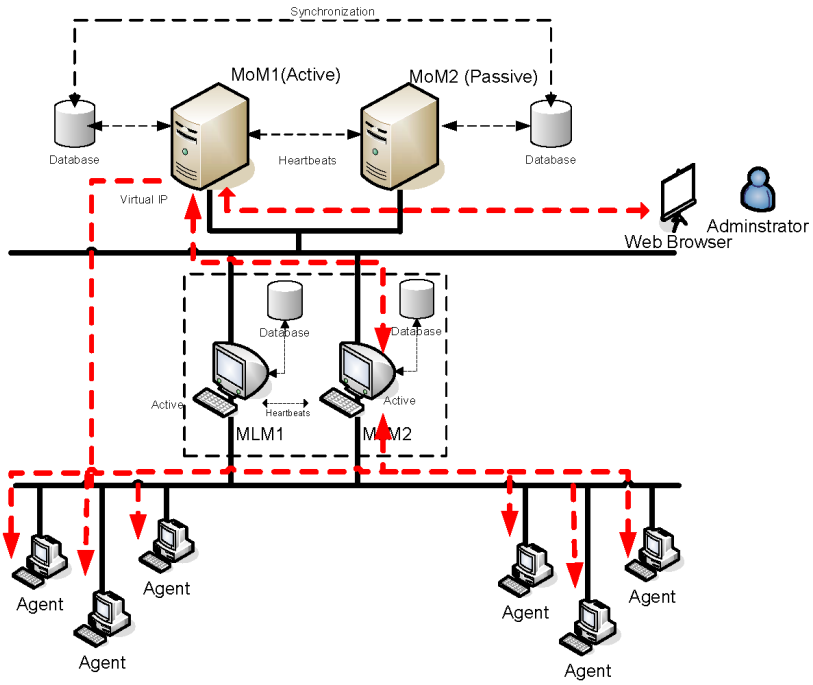


Figure 7.5: FTNMS: MLM1 is Down, MLM2 is Up.

Since MLMs are identical in the architecture, operation, and protocols, there are fewer cases compared to the MoM testing shown before.

When an MLM fails, its partner MLM will detect the failure after the deadtime specified in the configurations of the heartbeat. The partner MLM will assign to itself the IP address of the failed MLM. It will also take control of the shared image disk duplicating the database of the failed MLM. This situation is shown in Figure 7.5. The agents might have a small period of disconnectivity during the failover period. When the failed MLM recovers from the

failure, it restores control over its IP and disk.

Let us shed more lights on what might happen if an MLM fails in the middle of a certain operation. There are four cases to consider.

1. **When an MoM is sending an SNMP request to an MLM:** As with the MoM failure during an SNMP transaction, the MoM's request will timeout and it must be sent again, when the IP address has been mapped correctly to a working MLM.

2. **When an MLM is reporting a trap to the MoM:** The implementation of the MLM also logs any trap received from agents before it is forwarded to the MoM. When the trap is successfully delivered, it is marked accordingly. If by chance, an MLM fails after receiving a trap but before forwarding it to the MoM, the backup MLM will detect that and do it right after it takes over.

3. **When Receiving a Trap or a Response from an Agent:** If an MLM fails while receiving a trap or a response from an agent, or when an agent sends a trap or a response during the failover period, the SNMP operation will timeout if a certain error handling mechanism is implemented. This can be overcome by configuring the partner MLM as a secondary manager on the SNMP agent, then whenever the primary manager is not found, the secondary is contacted instead. The partner MLM has also been programmed to expect handle messages received from agents not belonging to it for such a case, see Figure 7.6.

4. **When an MLM is processing a request from the MoM:** This case was tested using Disman-Remop standard specifying how a management station can request an agent to ping certain nodes and report the results back. The standard was introduced in Section 2.1.

   The operation can be started by creating an entry in the pingCtlTable. In my implementation, whenever an MLM receives such a request, it logs the whole request and starts

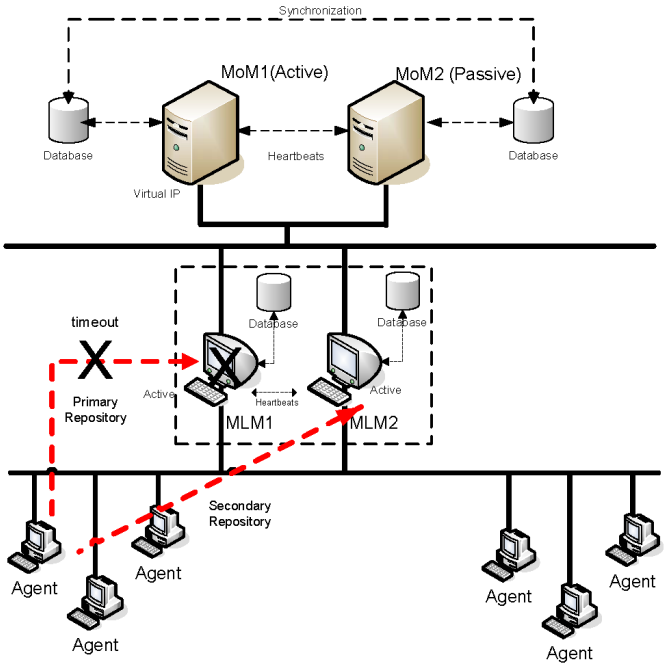


Figure 7.6: FTNMS: An Agents Reports to MLM1 While it is Down.

processing it. If it happens that the MLM failed before reporting the results through pingCtlNotification, the partner MLM will detect that, and then will process and report the result to the MoM.

Chapter 8

System Performance, Reliability, and Availability Analysis

8.1 Performance Evaluation

FTNMS provides reliability at other costs including **excess traffic**. In addition, the availability provided by FTNMS is not a continuous-availability but rather a high-availability with a downtime caused by the **failover delay**. In this section, we want to provide an estimation of both measures for the FTNMS system.

Throughout the remaining discussion, we will be using the following notation (FTNMS-n) to refer to our system where n is the number of pairs. For instance, FTNMS-1 refers to an FTNMS system with 1 pair of 2 MLMs.

8.1.1 Estimating the Excess Traffic

The bandwidth consumed by FTNMS will mainly come from two sources: the heartbeats exchange and the database synchronization.

The heartbeat traffic will depend on the frequency at which the heartbeats are sent and the

number of participating nodes. Hence, the bandwidth used by heartbeats ($\Delta B_{heartbeat}$) in bps is given by:

$$(\Delta B_{heartbeat}) = f \times L \times N \tag{8.1}$$

Where,

f = The frequency of heartbeats ($\frac{1}{\textit{Interval between heartbeats}}$).

L = The heartbeat messages size (in bits).

N = The number of participating nodes.

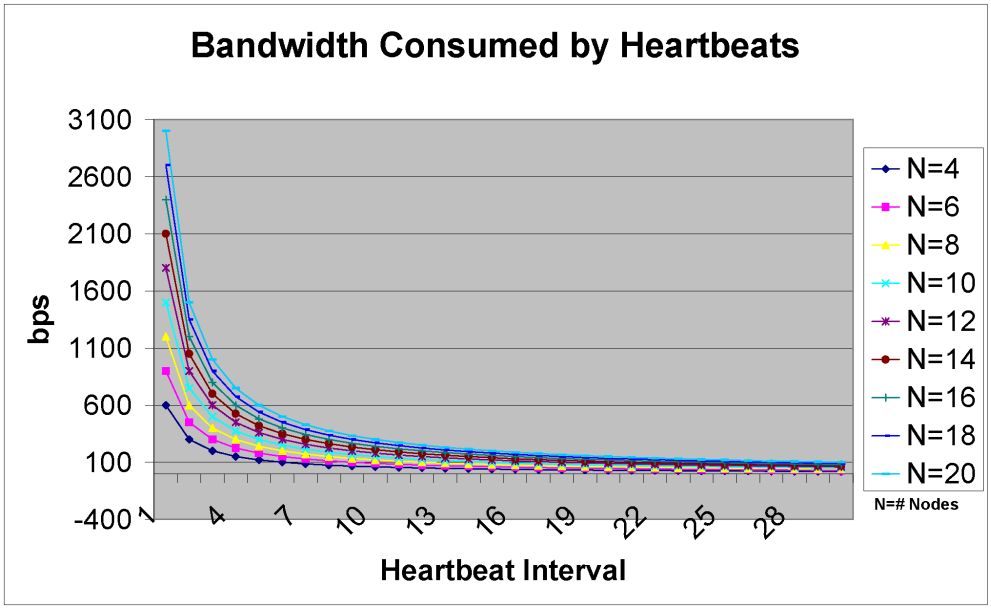


Figure 8.1: The Bandwidth Consumed by Heartbeats for N Nodes.

The heartbeat message size is about 150 bytes long [49]. Figure 8.1 shows how the bandwidth consumed by the heartbeats increases with the increase of the number of nodes. It can be inferred from the chart that even in the worst case with 1 heartbeat/sec, the bandwidth is less than 3Kbps. It is also interesting to note that for each new pairs of MLMs, the increase in bandwidth is of $2 \times 150 \times f$, which equals 300 bps in the worst case. Knowing that the

typical value for f is 0.1 meaning 1 heartbeat per 10 seconds, the heartbeat traffic can be of insignificant affect on the network bandwidth.

The major increase of traffic will however come from the database synchronization. This traffic will be moving between each pair of MLMs as well as from the active to the passive MoM. We can write an expression for the traffic (ΔB_{sync}) generated from synchronizing disks as:

$$\Delta B_{synch} = T_{MoM} + nT_{MLM}. \tag{8.2}$$

Where,

$T_{MoM} =$ The traffic generated by the active MoM.

$T_{MLM} =$ The traffic generated by each MLM.

$n =$ The number of MLMs.

Note that the MoM will have a one way traffic (from the active to the passive) whereas each pair of MLMs will be exchanging traffic both ways. It is clear from Equation 8.2 that the relation between the total bandwidth consumed by the synchronization process is linearly proportional to the number of MLMs.

For simplicity, we will assume that the traffic generated by an MoM equals that generated by an MLM. This assumption can be supported by the fact that although MLMs do more work such as polling and receiving traps, the MoM receives traffic from all MLMs. This makes it reasonable to assume that $T_{MoM} = T_{MLM} = T_{Node}$. We also assume that the time between two synchronizations is exponentially distributed with mean rate λ_{sync}. Furthermore, the amount of traffic sent during each synchronization can be assumed to follow a poisson distribution with mean $\lambda_{traffic}$.

We can now write an expression for the average traffic generated by MoMs and MLMs:

$$T_{Node} = \text{Average Traffic per Sync} \times \text{Sync Rate} = \lambda_{traffic}\lambda_{sync}.$$

Then the total traffic for n MLMs and one MoM can be written as:

$$\Delta B_{synch} = (n+1)\lambda_{traffic}\lambda_{sync}. \tag{8.3}$$

Figures 8.2, 8.3, 8.4, and 8.5 show the synchronization traffic for FTNMS-1, FTNMS-2, FTNMS-3, and FTNMS-4, respectively. The charts show that the synchronization will introduce a tremendous amount of traffic to the network. However, the following must be observed:

- The traffic must be considered with respect to the capacity of all pipes carrying the traffic not to the capacity of one link.
- The traffic will be confined within the subnet to which the MoMs or the MLMs belong. For instance, Figure 8.6 shows the expected traffic generated by the FTNMS-MoM for different synchronization periods. Under normal conditions, the traffic will not exceed 512 Kbps.
- The traffic will also be highly dependent on the number of agents monitored, the monitored objects (values to keep polling) and the polling interval.
- The way agents are distributed between MLMs will highly affect the amount of traffic generated.
- Since the normal polling interval of an MLM will be $>$ 1 minute, the normal size of the traffic per synchronization can be expected to be less that 128Kb.

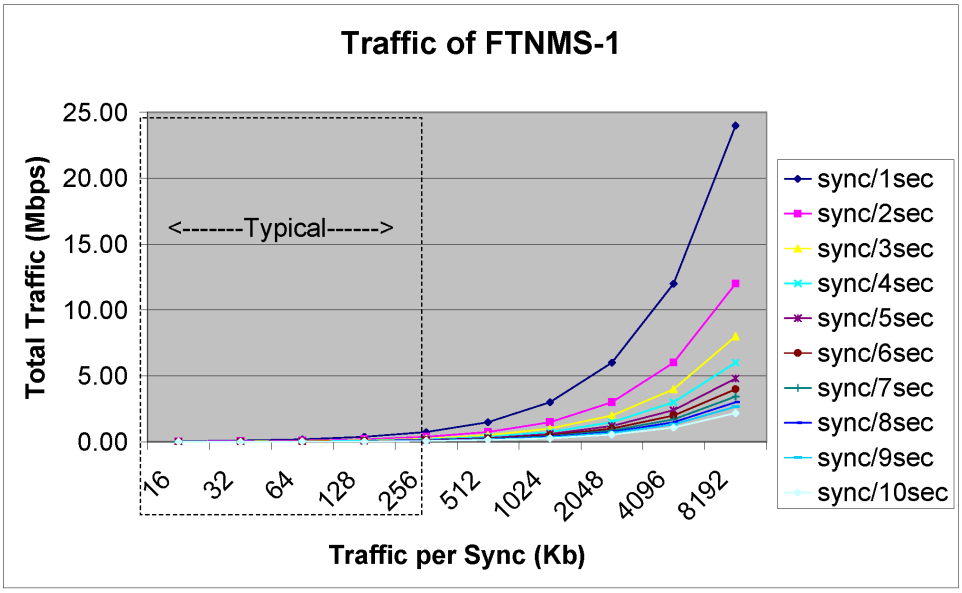


Figure 8.2: The Traffic Generated by FTNMS-1.

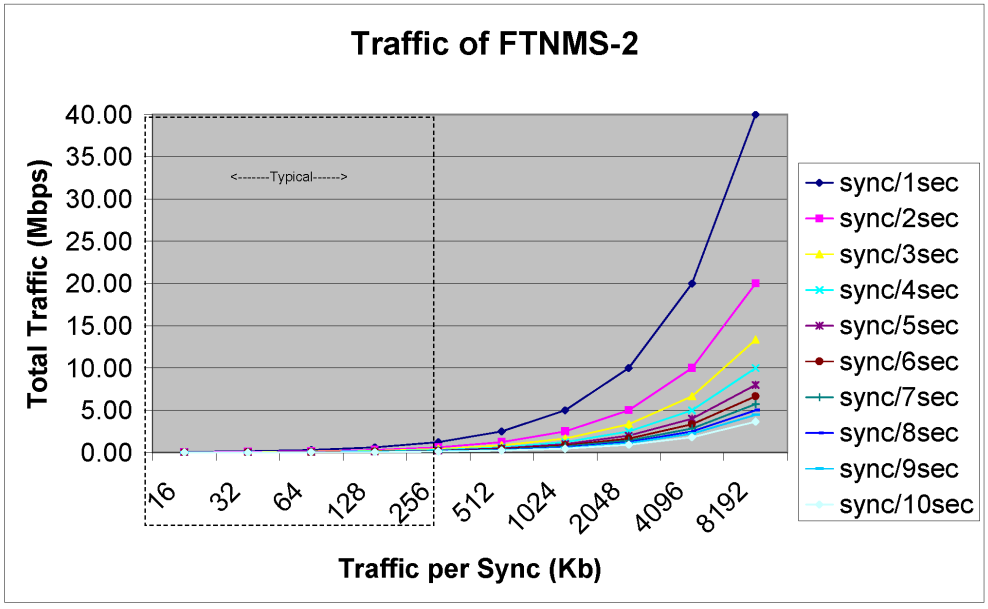


Figure 8.3: The Traffic Generated by FTNMS-2.

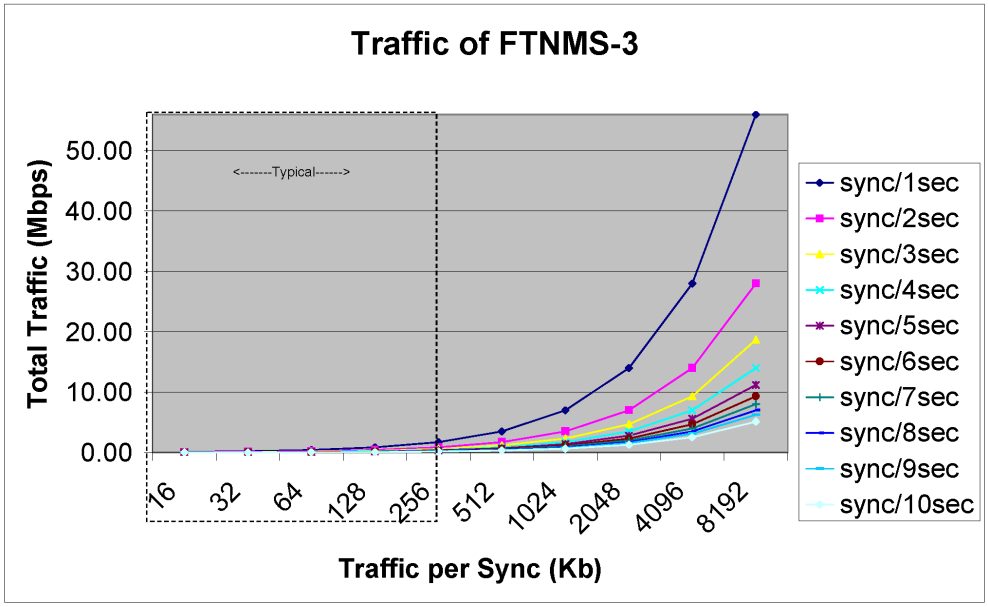


Figure 8.4: The Traffic Generated by FTNMS-3.

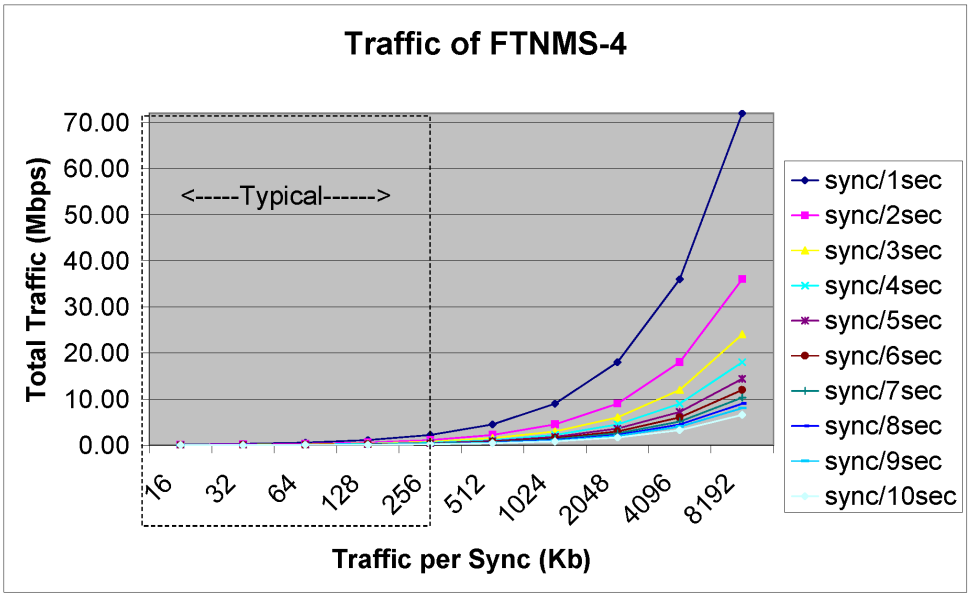


Figure 8.5: The Traffic Generated by FTNMS-4.

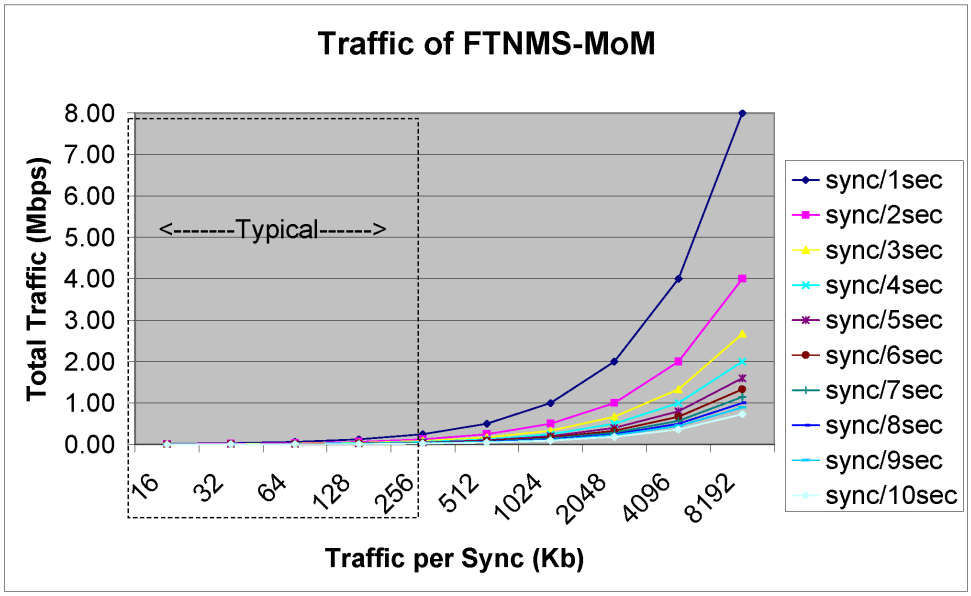


Figure 8.6: The Traffic Generated by FTNMS-MoM.

8.1.2 Failover Delay

The failover delay is a function of the **deadtime** which defines the interval after which a partner node is considered dead if no heartbeat is heard from it. If the deadtime is made too small, failure detection will be fast, but the probability of a false conclusion is higher since a heartbeat might get delayed by network congestion or the sending system being busy processing jobs. The higher the value, the slower the failure detection but the less probable the false conclusion is. The default value specified by Linux heartbeat designers is 10s.

The other factor that affects the failover process is the time taken by the ARP spoofing process to change the old IP/MAC mapping to the new one.

Hence, the failover delay (D) can be estimated as:

$$D = T_D + T_{arp} \tag{8.4}$$

Where,

$$T_D = \text{deadtime}$$
$$T_{arp} = \text{ARP spoofing time}$$

Since the system is implemented in a LAN environment, T_{arp} is going to be negligible compared to T_D. Hence we will ignore the effect of T_{arp}. The delay can be represented by a straight line as in Figure 8.7.

8.2 Reliability Estimation

To evaluate the reliability of the FTNMS, we will start by modeling the system shown in Figure 8.8 as an RBD. Then we will develop an expression to describe the reliability of the system. The equivalent RBD for the system is shown in Figure 8.9

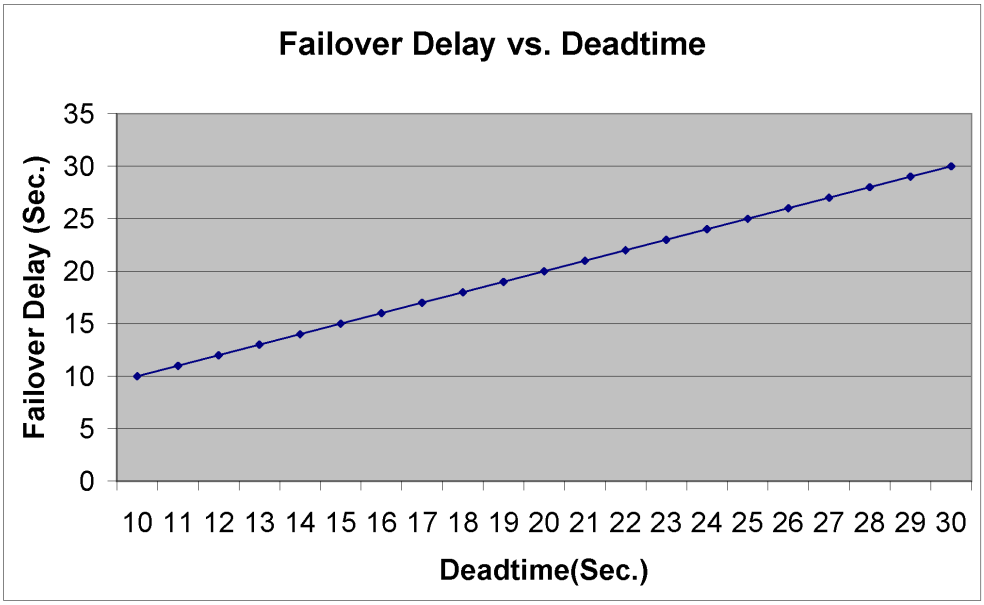


Figure 8.7: Failover Time vs. deadtime.

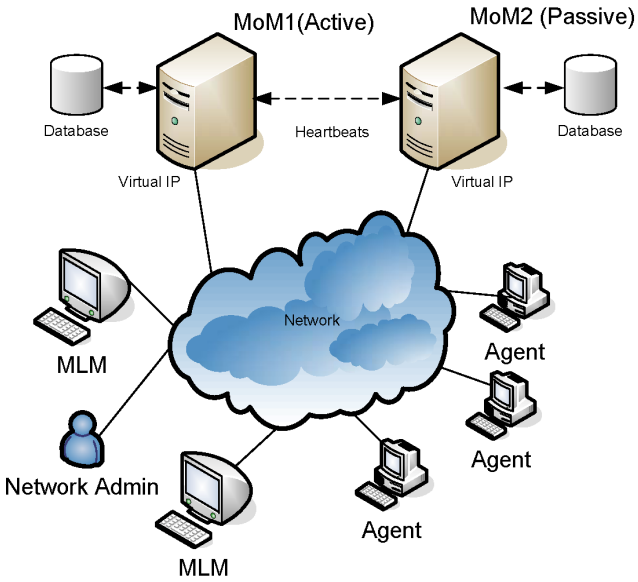


Figure 8.8: The Arrangement of the Proposed Network Management System.

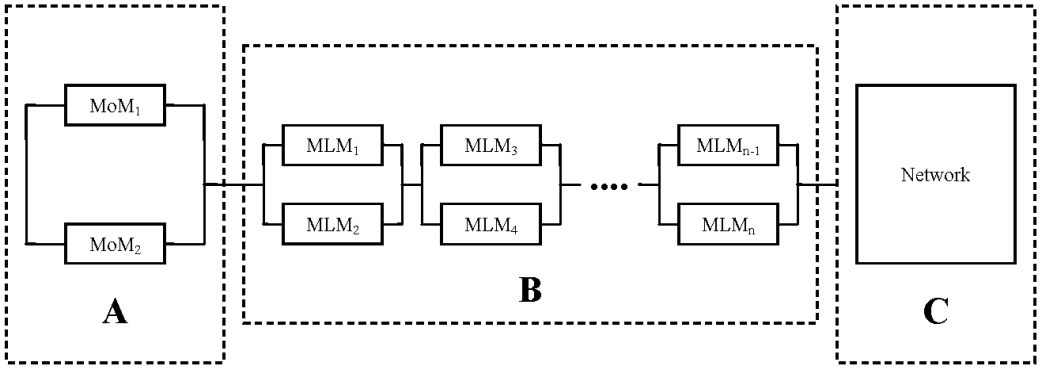


Figure 8.9: The RBD of the Proposed Design

It is important to note that the following are assumed for the model we are dealing with:

- The components are independent, meaning that the failure of any component has no effect on the reliability of the other components.
- The failure of at least one pair of MLMs will be considered a failure of the whole system.
- The system will have no impact on the reliability of the network. Moreover, the network will be modeled as one component with its own reliability (R_N). This assumption aims at considering the network reliability in the overall model, without adding complexity to it.
- The reliability of the nodes includes the reliability of the connecting links, i.e., the failure of the link attached to the node is going to be considered a failure of the node. In other words, R_{node} includes R_{links}.

The strategy is to divide the system into three parts: A, B and C to simplify the analysis. Part A represents the MoM layer, Part B represents the MLM layer and Part C represents the rest of the network.

Now, let

$R(FTNMS) =$ The reliability of the FTNMS system

$R(FTNMS_{MoM}) =$ The reliability of the FTNMS-MoM part

$R(FTNMS_{MLM}) =$ The reliability of the FTNMS-MLM part

$R_N =$ The reliability of the network

Since the system in Figure 8.9 is a series system, we can express $R(FTNMS)$ as,

$$R(FTNMS) = R(FTNMS_{MoM}) \times R(FTNMS_{MLM}) \times R_N \tag{8.5}$$

We will now calculate the expression for both $R(FTNMS_{MoM})$ and $R(FTNMS_{MLM})$ separately.

8.2.1 Reliability Analysis for FTNMS-MoM

Figure 8.10 shows an RBD for the FTNMS-MoM. It can be seen from the RBD that the FTNMS-MoM is nothing but a simple parallel system. Therefore, its reliability can be expressed as:

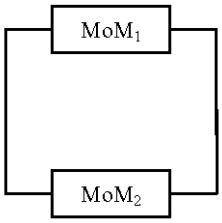


Figure 8.10: The RBD of the MoM Section.

$$R(FTNMS_{MoM}) = 1 - (1 - R_{MoM1})(1 - R_{MoM2})$$

Where,

R_{MoM1} =Reliability of MoM_1

R_{MoM2} =Reliability of MoM_2

If we assume that $R_{MoM1} = R_{MoM2} = R_{MoM}$ meaning the reliability of both MoMs is equal, then we can rewrite the FTNMS-MoM reliability equation as:

$$R(FTNMS_{MoM}) = 2R_{MoM} - R_{MoM}^2$$

The FTNMS-MoM can be used alone to improve the reliability of centralized systems. For that reason, we will compare it with a single NMS system to see the improvement we get in reliability when using FTNMS-MoM instead of an ordinary single NMS system. Let the reliability of a single NMS system be R_{NMS}. If we assume that $R_{MoM} = R_{NMS}$, then the gain in reliability of an FTNMS-MoM over a single NMS system is given by

$$R_{gain} = \frac{(2R_{NMS} - R_{NMS}^2) - (R_{NMS})}{R_{NMS}} \times 100\% = 100 \times (1 - R_{NMS})\%$$

Figure 8.11 shows the reliability of both systems for the same value of R_{NMS}. The figure shows clearly how the reliability of the network management system improves when using FTNMS-MoM.

To further assess the reliability improvement, we can calculate the expected value of both reliabilities. The average value of the reliability of FTNMS-MoM and centralized NMS system can be expressed as:

$$E(R_{FTNMS-MoM}) = \int_0^1 2R - R^2 \, dR = R^2 - \frac{R^3}{3} = 0.67$$

$$E(R_{NMS}) = \int_0^1 R \, dR = \frac{R^2}{2} = 0.5$$

This means that if the FTNMS-MoM was used alone to build a centralized NMS, it will result in more reliability compared to that of an ordinary one.

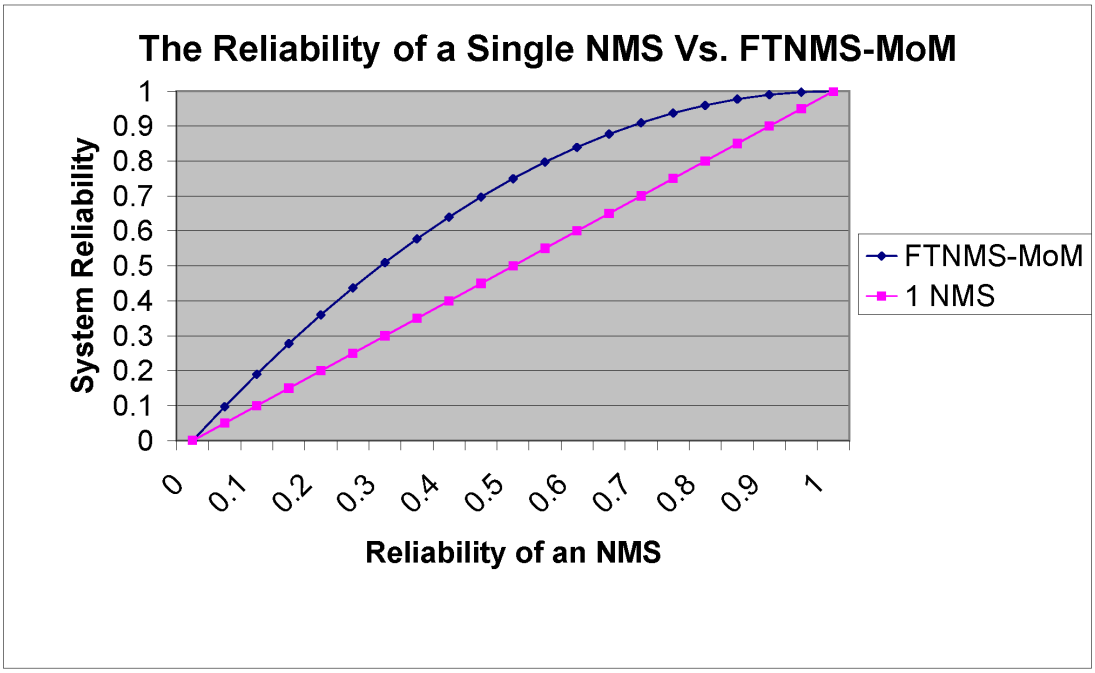


Figure 8.11: The Reliability of a Single-NMS System vs. FTNMS-MoM System.

8.2.2 Reliability Analysis for FTNMS-MLM

An RBD for the FTNMS-MLM part is shown in the Figure 8.12. It can be seen from the diagram that for FTNMS-MLM to fail, one pair of MLMs has to fail. Additionally, a pair will only fail when both MLMs comprising it fail. Assuming R_{MLM_n} to be the reliability of MLM_n, we can express the of reliability of the n^{th} pair as:

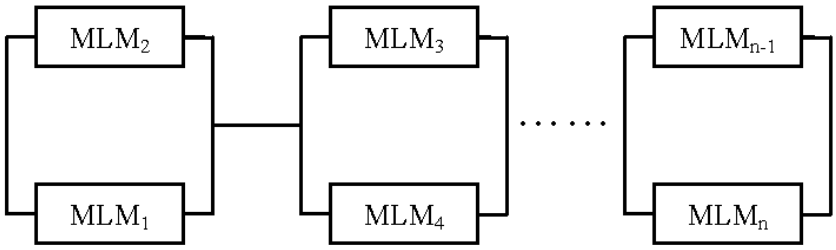


Figure 8.12: RBD for FTNMS-MLM.

$$R(MLM_n) = R_{2n-1} + R_{2n} - R_{2n-1}R_{2n}.$$

Then, for m pairs,

$$R(FTNMS_{MLM}) = \prod_{n=1}^{m}(R_{2n-1} + R_{2n} - R_{2n-1}R_{2n}).$$

If R_{MLM_n} for all MLMs is equal, then:

$$R(FTNMS_{MLM}) = (2R_{MLM} - R_{MLM}^2)^m$$

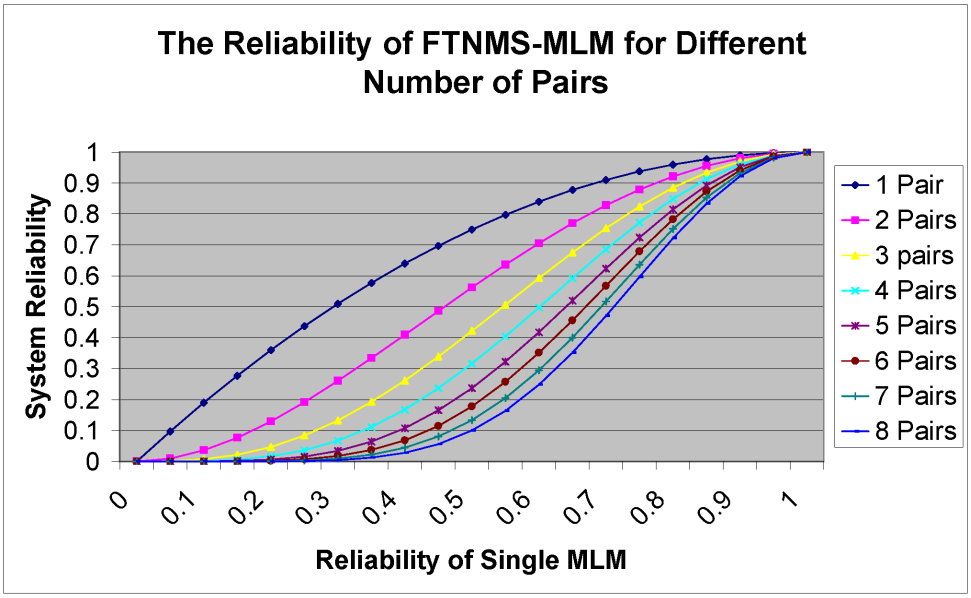


Figure 8.13: The Reliability of FTNMS-MLM for Different Number of Pairs.

Where m is the number of pairs of MLMs. Figure 8.13 shows how the reliability of the whole MLM is affected by the increase of MLM pairs. Since we assumed that MLM pairs are working in series, it is expected that the overall reliability will decrease as the number of

MLM pairs increases.

8.2.3 The Overall Reliability of FTNMS

The overall reliability can now be obtained by substituting the values of $R(FTNMS_{MoM})$ and $R(FTNMS_{MLM})$ into Equation 8.5. Hence,

$$R(FTNMS) = (2R_{MoM} - R_{MoM}^2) \times (2R_{MLM} - R_{MLM}^2)^m \times R_N$$

Where,

$$R_{MoM} = \text{The reliability of an MoM.}$$
$$R_{MLM} = \text{The reliability of an MLM.}$$
$$m = \text{Number of MLM pairs.}$$
$$R_N = \text{Network reliability.}$$

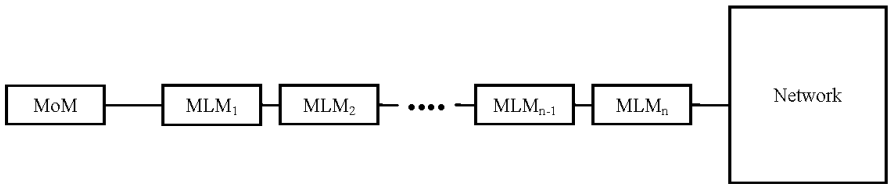


Figure 8.14: An RBD for a Typical Hierarchical NMS System.

Let's compare this reliability with the reliability of the ordinary hierarchical NMS system shown in Figure 8.14. Using the same strategy as with FTNMS, we can express the reliability of that system as:

$$R(NMS_{Hierarchical}) = (R_{MoM}) \times (R_{MLM})^n \times R_N$$

Where,

R_{MoM} = The reliability of the MoM.

R_{MLM} = The reliability of an MLM.

n = Number of MLMs.

R_N = Network reliability.

For a fair comparison, $m = n/2$. Meaning that the number of MLMs for the hierarchical system and FTNMS under comparison must be the same.

Figures 8.15, 8.16, 8.17 and 8.18 show the reliability of FTNMS versus a hierarchical system for different number of MLMs. Note, that the notation Hierarchical-*n* refers to a Hierarchical NMS with *n* MLMs. In the figure, we assume that MoMs of FTNMS and the MoM of the hierarchical NMS have the same reliability (R_{MoM}). We further assume that the MLMs of both systems will also have the same reliability (R_{MLM}).

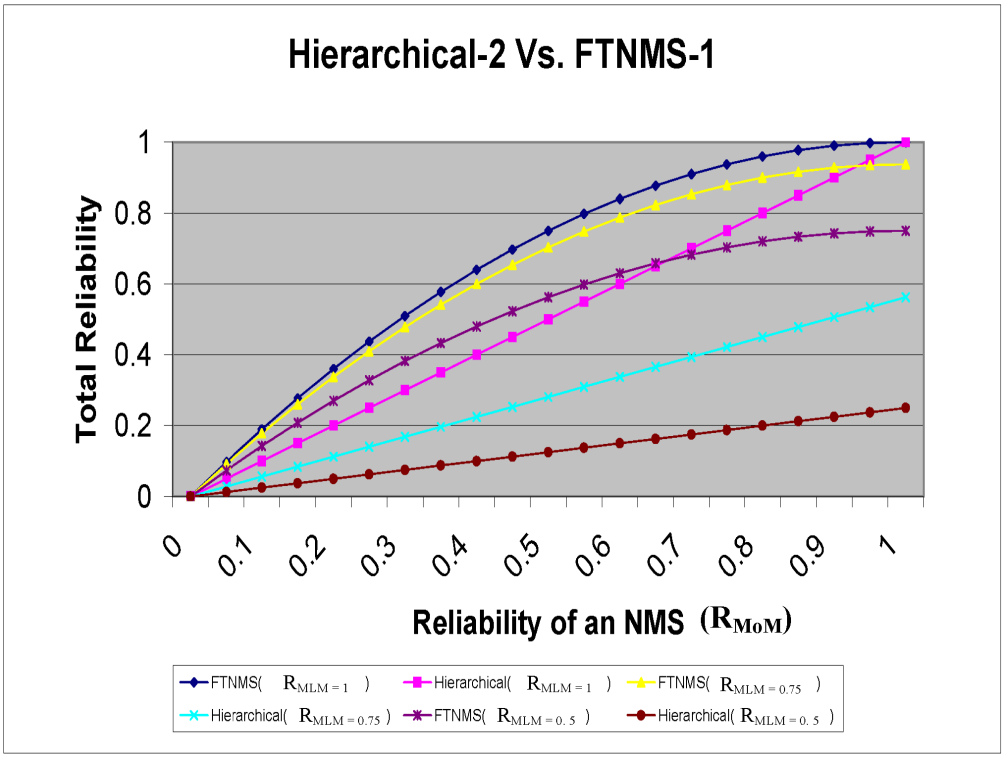


Figure 8.15: The Reliability of a Hierarchical-2 Vs. FTNMS-1.

To be consistent, curves with similar values of R_{MLM} must be compared. For example, if one was to compare the FTNMS with $(R_{MLM} = 0.25)$, it must be compared with the Hierarchical system with $(R_{MLM} = 0.25)$. The following results can be deduced from the charts:

- With the decrease in the reliability (R_{MoM} & R_{MLM}), the total reliability of the system decreases drastically for the ordinary hierarchical systems but the FTNMS sustains a very good level of reliability even with high failure rates.
- The more MLMs are in the system, the less the total reliability is. This is due to the fact that MLMs and MLM pairs are assumed to be working in series. Therefore, the more sequential components we add to the system, the more its probability of failure.
- Since today's hardware and software reliability is high, we do not need to worry about the reliability decreasing with the number for MLMs since the FTNMS keeps a very good level of reliability with low failure rates.
- It must be also noted that in addition to reliability, FTNMS increases the fault-tolerance and the availability of NMSs. The latter provides another criteria to judge the design.

The gain in reliability resulted from using the FTNMS system can be calculated as

$$\begin{aligned} R_{gain} &= \frac{R_{FTNMS} - R_{Hierarchical}}{R_{Hierarchical}} \\ &= \frac{(2 - R_{MoM})(2 - R_{MLM})^m - R_{MLM}^m}{R_{MLM}^m} \\ &= (2 - R_{MoM})(\frac{2}{R_{MLM}} - 1)^m - 1 \end{aligned}$$

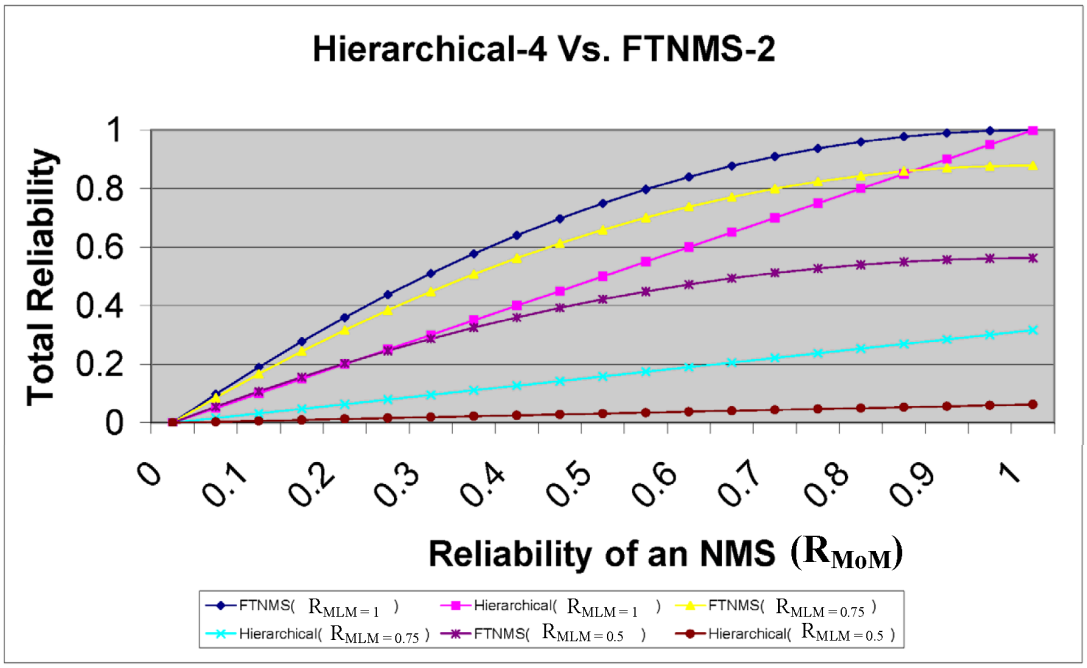


Figure 8.16: The Reliability of a Hierarchical-4 Vs. FTNMS-2.

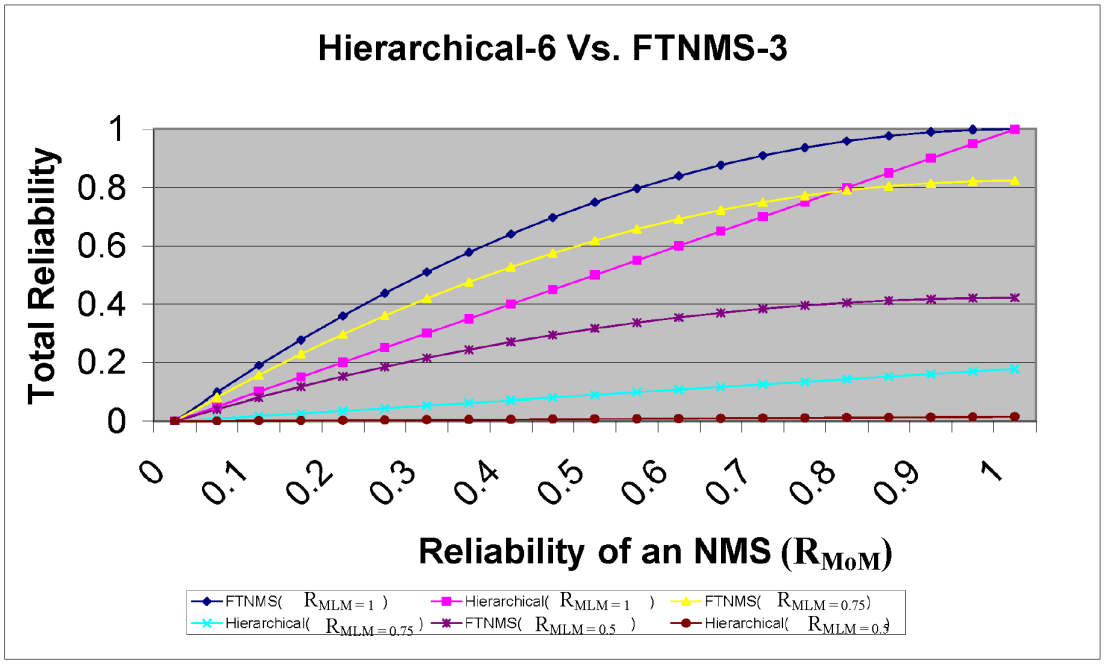


Figure 8.17: The Reliability of a Hierarchical-6 Vs. FTNMS-3.

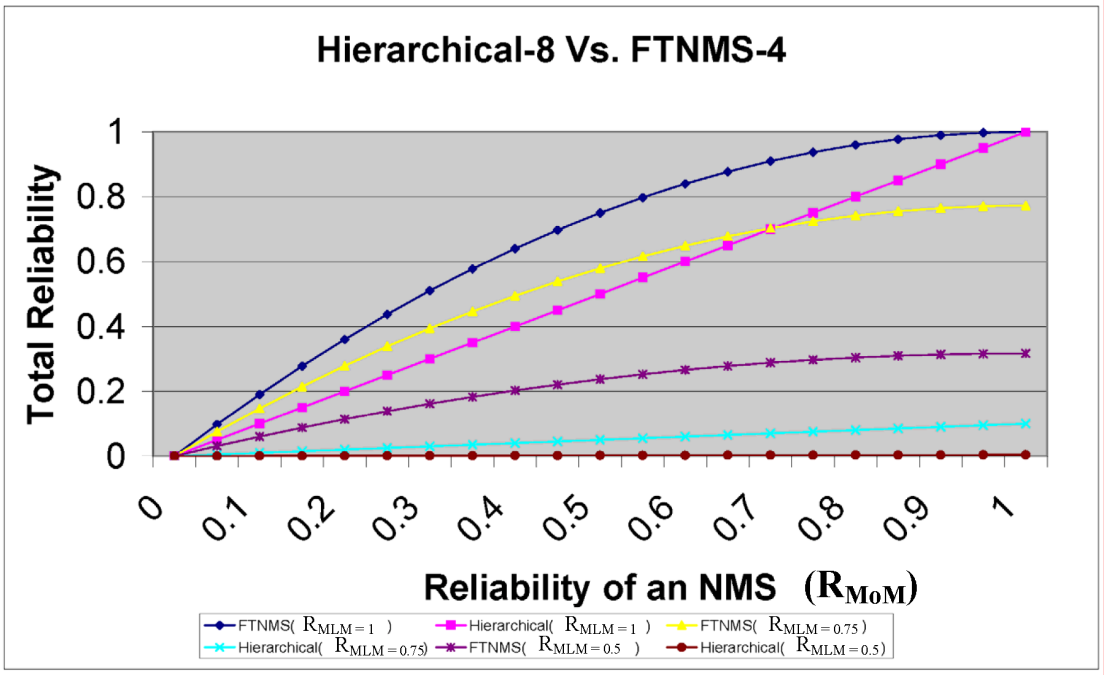


Figure 8.18: The Reliability of a Hierarchical-8 Vs. FTNMS-4.

8.3 Availability Estimation

Availability in a client-server environment is treated differently from other systems due to the fact that the failure of a server or a terminal are considered both as unavailability of the system [42]. For the purpose of this study however, we will assume the following:

- The system is considered available only when at least one MoM is up and each pair of MLMs has at least one MLM up. This implies that if we have both MoMs down or at least one pair of MLMs with two failed MLMs, the system is considered unavailable.
- Only one transition can happen at a time. Meaning no two failures or recoveries can happen simultaneously.
- The system is repairable.
- The failure and repair rates are constant.
- To simplify the argument, we will assume that all MoMs and MLMs have the same failure rate λ and the same repair rate μ.

Let's now start the analysis:

1. The system states will be represented as a sequence of digits of the form $Sx_0x_1..x_m$. The left-most digit (x_0) represents the number of working MoMs. The next digits represent the number of working MLM in pairs $1, 2, .., m$. All digits can have values between 0 and 2. For instance, for a system with 2 pairs of MLMs where 1 MLM is down in pair#1, we represent this state as $S212$. Similarly, if one MoM is down and both MLMs in pair#2 are down, the state is represented as $S120$.
2. A Markov chain representation of FTNMS-1 and its states transitions is shown in Figure 8.19. Note that the transition from a state to itself is not shown but it can be

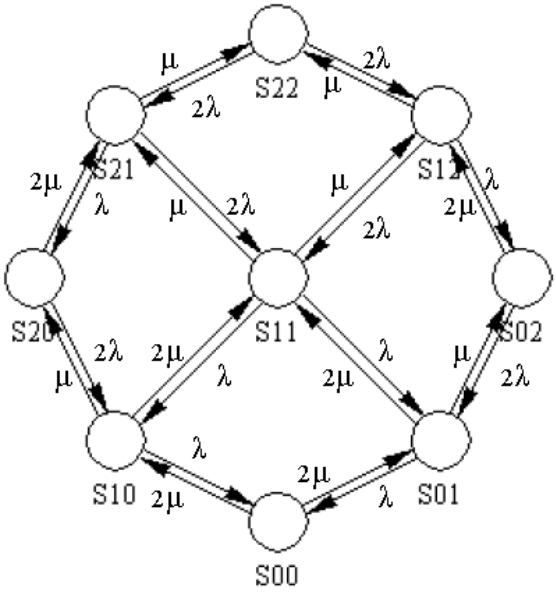


Figure 8.19: A Markov Chain of FTNMS-1.

calculated as the negative of the total rates going out of the state. For example, we can move from $S22$ to $S21$ with probability of 2λ since either MLM can fail. However, moving from $S21$ to $S22$ has a probability μ since only one MLM is failed and it is the only MLM to recover.

3. Table 8.1 shows all the transitions happening between each two states. Notice, that the transition rates from states to themselves have also been included. For example, staying in $S21$ has a probability of $3\lambda + \mu$. This expression is due to the fact that the probability of staying in $S21$ equals to the probability of not moving from $S21$. This probability can be expressed by summing the probabilities of all transitions going out of $S21$.

4. The transition matrix corresponding to our system will be called $T_{FTNMS-1}$. The transition matrix was built using the transition table. Note that the probability of staying in the same state has been negated.

Table 8.1: State Transitions for FTNMS-1

From | To	S_{22}	S_{12}	S_{21}	S_{11}	S_{02}	S_{20}	S_{10}	S_{01}	S_{00}
S_{22}	4λ	2λ	2λ	0	0	0	0	0	0
S_{12}	μ	$3\lambda+\mu$	0	2λ	λ	0	0	0	0
S_{21}	μ	0	$3\lambda+\mu$	2λ	0	λ	0	0	0
S_{11}	0	μ	μ	$2(\lambda+\mu)$	0	0	λ	λ	0
S_{02}	0	2μ	0	0	$2(\lambda+\mu)$	0	0	2λ	0
S_{20}	0	0	2μ	0	0	$2(\lambda+\mu)$	2λ	0	0
S_{10}	0	0	0	2μ	0	μ	$\lambda+3\mu$	0	λ
S_{01}	0	0	0	2μ	μ	0	0	$\lambda+3\mu$	λ
S_{00}	0	0	0	0	0	0	2μ	2μ	4μ

$$T_{FTNMS-1} = \begin{bmatrix} -(4\lambda) & 2\lambda & 2\lambda & 0 & 0 & 0 & 0 & 0 & 0 \\ \mu & -(3\lambda+\mu) & 0 & 2\lambda & \lambda & 0 & 0 & 0 & 0 \\ \mu & 0 & -(3\lambda+\mu) & 2\lambda & 0 & \lambda & 0 & 0 & 0 \\ 0 & \mu & \mu & -2(\lambda+\mu) & 0 & 0 & \lambda & \lambda & 0 \\ 0 & 2\mu & 0 & 0 & -2(\lambda+\mu) & 0 & 0 & 2\lambda & 0 \\ 0 & 0 & 2\mu & 0 & 0 & -2(\lambda+\mu) & 2\lambda & 0 & 0 \\ 0 & 0 & 0 & 2\mu & 0 & \mu & -(\lambda+3\mu) & 0 & \lambda \\ 0 & 0 & 0 & 2\mu & \mu & 0 & 0 & -(\lambda+3\mu) & \lambda \\ 0 & 0 & 0 & 0 & 0 & 0 & 2\mu & 2\mu & -(4\mu) \end{bmatrix}$$

$$P_{FTNMS-1} = \begin{bmatrix} P_{22} \\ P_{12} \\ P_{21} \\ P_{11} \\ P_{02} \\ P_{20} \\ P_{10} \\ P_{01} \\ P_{00} \end{bmatrix} \quad O = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

5. Now, we can derive the equations for all the states using $T_{FTNMS-1} \times P_{FTNMS-1} = O$

$$
\begin{aligned}
&-(4\lambda)P_{22} + \mu P_{12} + \mu P_{21} = 0 \\
&2\lambda P_{22} - (3\lambda + \mu)P_{12} + \mu P_{11} + 2\mu P_{02} = 0 \\
&2\lambda P_{22} - (3\lambda + \mu)P_{21} + \mu P_{11} + 2\mu P_{20} = 0 \\
&2\lambda P_{12} + 2\lambda P_{21} - 2(\lambda + \mu)P_{11} + 2\mu P_{10} + 2\mu P_{01} = 0 \\
&\lambda P_{12} - 2(\lambda + \mu)P_{02} + \mu P_{01} = 0 \\
&\lambda P_{21} - 2(\lambda + \mu)P_{20} + \mu P_{10} = 0 \\
&\lambda P_{11} + 2\lambda P_{02} - (\lambda + 3\mu)P_{01} - 2\mu P_{00} = 0 \\
&\lambda P_{11} + 2\lambda P_{20} - (\lambda + 3\mu)P_{10} - 2\mu P_{00} = 0 \\
&\lambda P_{10} + \lambda P_{01} - (4\mu)P_{00} = 0
\end{aligned}
$$

6. We can solve the system of equations above for the probabilities of the states which represents the fault-free state of the systems, namely $P_{22}, P_{12}, P_{21}, P_{11}$. The solution can be found using the above system of equations in addition to using the equation:

$$P_{22} + P_{12} + P_{21} + P_{11} + P_{02} + P_{20} + P_{01} + P_{10} + P_{00} = 1.$$

Solving the equations using Matlab yields:

$$A_{FTNMS-1} = P_{22} + P_{12} + P_{21} + P_{11} = \frac{\mu^4 + 4\lambda\mu^3 + 4\lambda^2\mu^2}{\mu^4 + \lambda^4 + 4(\lambda^3\mu + \lambda\mu^3) + 6\lambda^2\mu^2}$$

For a fair comparison, we will also derive the availability of a Hierarchal system with the same number of MLMs using the same approach. Figure 8.20 shows the Markov chain of a typical Hierarchical-2. Similarly, Table 8.2 shows the transition table.

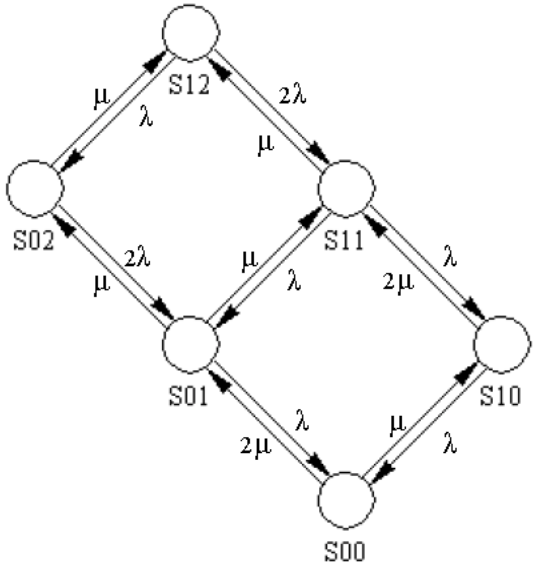


Figure 8.20: A Markov Chain of A Hierarchical NMS with 2MLMs.

$$\begin{bmatrix} -(3\lambda) & \lambda & 2\lambda & 0 & 0 & 0 \\ \mu & -(2\lambda+\mu) & 0 & 0 & 2\lambda & 0 \\ \mu & 0 & -(2\lambda+\mu) & \lambda & \lambda & 0 \\ 0 & 0 & 2\mu & -(\lambda+2\mu) & 0 & \lambda \\ 0 & \mu & \mu & 0 & -(\lambda+2\mu) & \lambda \\ 0 & 0 & 0 & \mu & 2\mu & -(3\mu) \end{bmatrix} \times \begin{bmatrix} P_{12} \\ P_{02} \\ P_{11} \\ P_{10} \\ P_{01} \\ P_{00} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Table 8.2: Transition Table for an Ordinary Hierarchical NMS with 2 MLMs.

From | To	S_{12}	S_{02}	S_{11}	S_{10}	S_{01}	S_{00}
S_{12}	3λ	λ	2λ	0	0	0
S_{02}	μ	$2\lambda+\mu$	0	0	2λ	0
S_{11}	μ	0	$2\lambda+\mu$	λ	λ	0
S_{10}	0	0	2μ	$\lambda+2\mu$	0	λ
S_{01}	0	μ	μ	0	$\lambda+2\mu$	λ
S_{00}	0	0	0	μ	2μ	3μ

$$-(3\lambda)P_{12} + \mu P_{11} + \mu P_{02} = 0$$
$$\lambda P_{12} - (2\lambda + \mu)P_{02} + \mu P_{01} = 0$$
$$2\lambda P_{12} - (2\lambda + \mu)P_{11} + 2\mu P_{10} + \mu P_{01} = 0$$
$$\lambda P_{11} - (\lambda + 2\mu)P_{10} + \mu P_{00} = 0$$
$$2\lambda P_{02} + \lambda P_{11} - (\lambda + 2\mu)P_{01} + 2\mu P_{00} = 0$$
$$\lambda P_{10} + \lambda P_{01} - (3\mu)P_{00} = 0$$

Solving the above system of equations, we get the following expression for the availability of Hierarchical-2.

$$A_{Hierarchical-2} = P_{12} = \frac{\mu^3}{3(\lambda^2\mu + \lambda\mu^2) + \lambda^3 + \mu^3}$$

A pictorial comparison of the two systems can be seen in the chart in Figure 8.21. The figure was produced using the equations for $A_{FTNMS-1}$ and $A_{Hierarchical-2}$. To verify the analysis, Figure 8.22 was produced using a reliability analysis tool called SHARPE [50]. The tool uses the markov chain of the system to produce its availability graph.

The figure shows curves for different values of μ. The figure shows clearly how the FTNMS-1 outperforms its equivalent hierarchical system.

Due to the complication in deriving the availability of a FTNMS-n with multiple pairs of MLMs ($n > 1$), we used SHARPE to produce FTNMS-2 availability. Figure 8.23 shows the availability of FTNMS-2 versus that of the Hierarchical-4 for different values of μ. The code used to produce the graph can be found in Appendix C.

The charts show that even when the reliability of the system components is high, the difference in availability of both systems is far higher with FTNMS. For example, when $\lambda = 0.9$, FTNMS-1 achieves $A \simeq 0.98$, while Hierarchical-2 has $A \simeq 0.72$. The situation is even better when comparing FTNMS-2 to Hierarchical-4, where the former has $A \simeq 0.98$ whereas the latter drops to $A \simeq 0.63$.

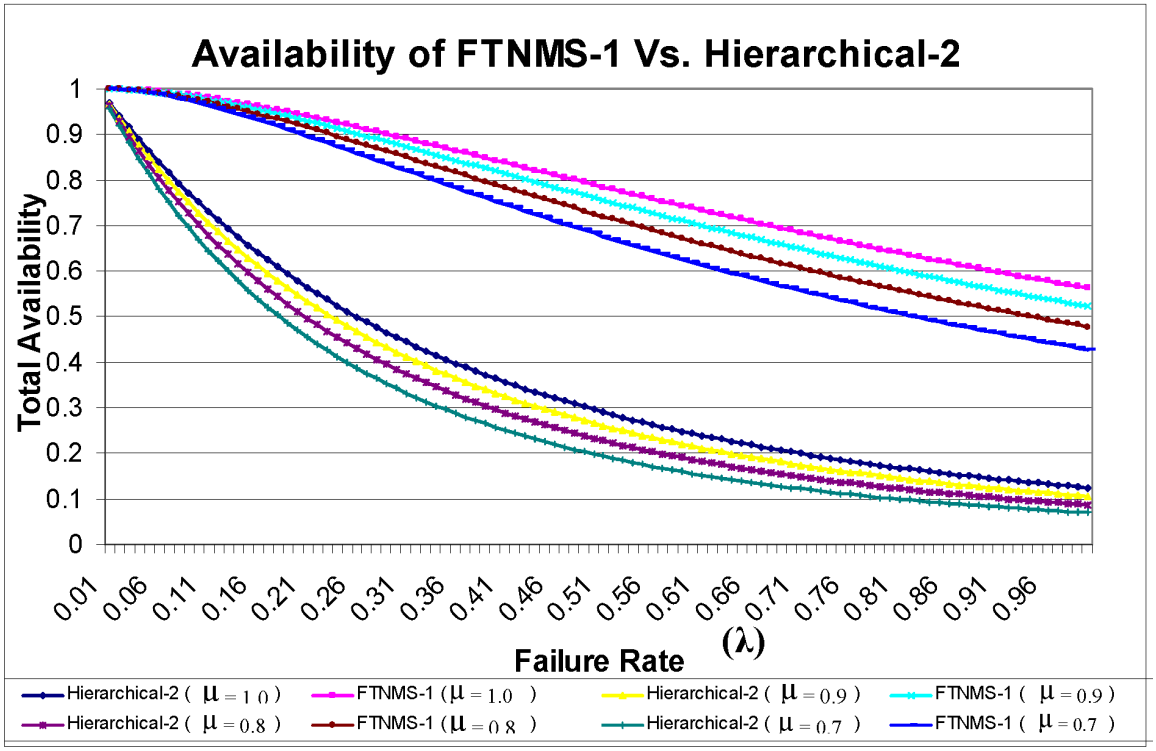


Figure 8.21: Availability: FTNMS-1 vs. Hierarchical-2.

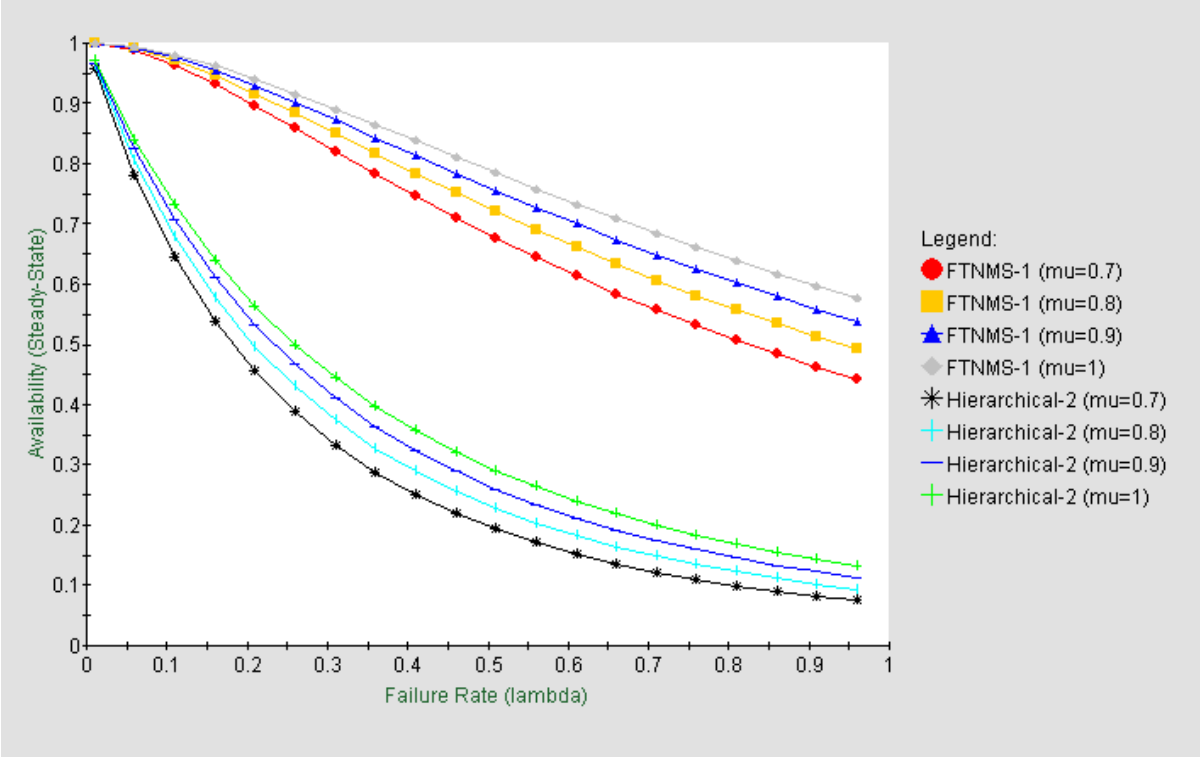


Figure 8.22: Availability: FTNMS-1 vs. Hierarchical-2 Using SHARPE.

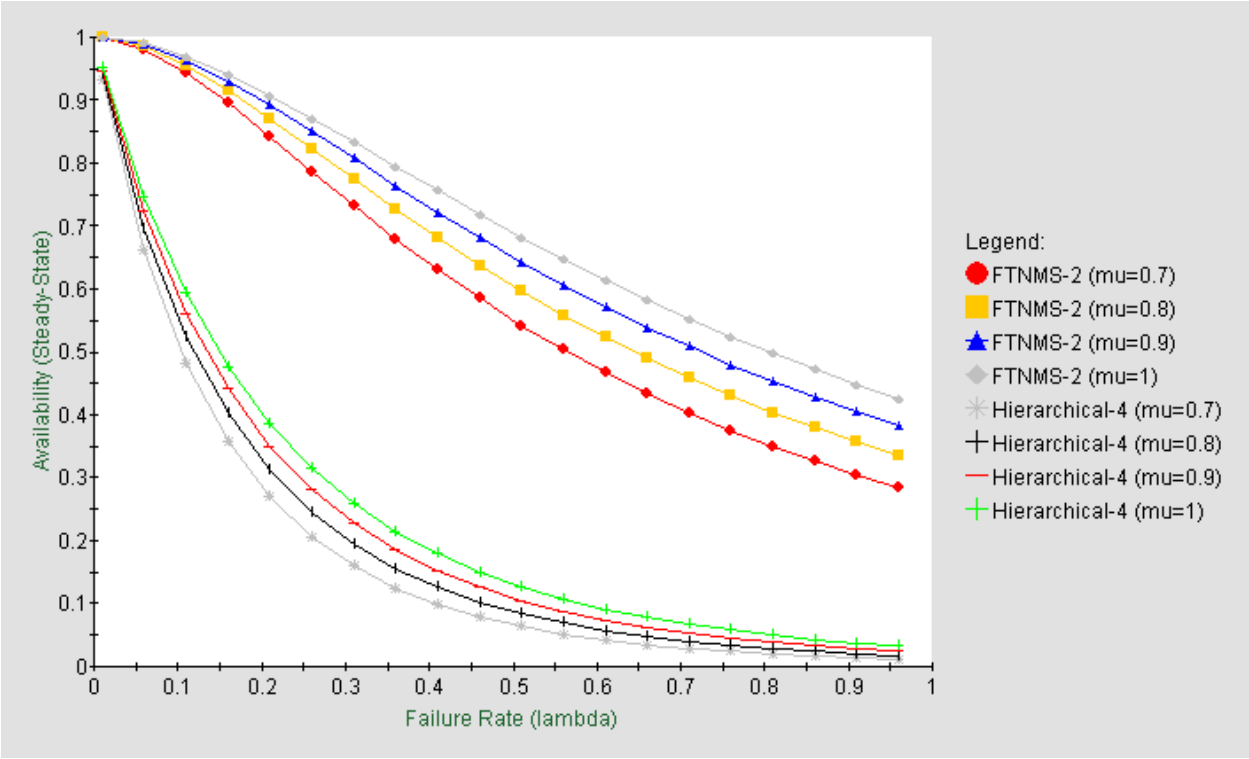


Figure 8.23: Availability: FTNMS-2 vs. Hierarchical-4

8.4 Comparing FTNMS with Other Systems

The evaluation conducted in this chapter shows that FTNMS can greatly improve the reliability and availability of NMSs. However, a little cost will appear in the extra traffic used for heartbeats and database synchronization which is typical for any fault-tolerant system. For example, when a centralized system is used with an NMS that has a reliability R_{NMS}, the NMS can increase the reliability by a factor of $1 - R_{NMS}$. The availability will improve by 0.26-0.35 for systems with two or four MLMs.

In this section, FTNMS will be compared with other systems surveyed in the related work. The comparison shows clearly that FTNMS has many strong points compared to its competitors. One of these is that its building blocks are off-the-shelf components, meaning that these

	Hi-ADSD	NetKeeper	RNMP	FTNMS
Paradigm	Distributed	Distributed	Hierarchical	Hierarchical/Centralized
Technology	Uses HI-ADSD Algorithm, SNMP	Distributed-Object , SNMP	New agent and RB, SNMP, CMIP	Linux, SNMP, Clustering
Agents	New	New	Ordinary SNMP or CMIP Agents	Ordinary SNMP Agents
Manager	No (Fully Distributed)	New	New	Ordinary Manager
Overhead	Inter-Agents Communication	Monitoring Servers, Hardware	MD, RP	Heart beats, Redundant Hardware
Failure Detection	HI-ADSD Algorithm	Distributed-Object Tool	No detection	Heartbeats
Database Synchronization	Not Mentioned	Not Mentioned	No need	DRBD
Agent-Agent Agent-Manager Communication	SNMP	Message Passing, RPC, Socket	RPC, IPC	SNMP
Claims	Can run even if only one agent is fault-free	Scalable, Fault-Resistant & Customizable	Works with both SNMP & CMIP	Easy Configuration, Ordinary tools, independent from Mgmt Protocol & Mgmt tool used

Table 8.3: A Comparison Between FTNMS & Other Systems.

components are all widely used and many of them are standards. The building blocks include Linux, SNMP, DRBD, and heartbeats. In addition, FTNMS suggests a mechanism for solving the database synchronization between nodes without losing the unified view of the system. Table 8.3 compares FTNMS with other approaches on different aspects.

The RNMP is not meant for reliability, rather it mainly improves interpretability. The only aspect of fault-tolerance in its design is when the RB or MD is down, the manager communicating with them will try to find another MD or RB. The design does not specify any other reliability aspect that can be compared with FTNMS.

The Hi-ADSD is good in that it is a distributed system. However, its implementation requires new agents to be implemented.

The NetKeeper uses distributed objects technology which hides a lot of details from the managers such as addressing, failures, etc. On the other hand, it requires a new manager and agents. Another disadvantage is the fact that the distributed object technology on which the NetKeeper is built is expensive.

It is to be noted that Hi-ADSD and NetKeeper address only the failure detection and some failover details but do not discuss the database synchronization and how the backup node can

continue the service on behalf of the failed one. In addition, the studies do not discuss the different scenarios that might happen when the failure happens while a transaction is taking place.

The FTNMS on the other hand, provides a realistic solution that is affordable and can be implemented easily. It can also fit with an existing NMS seamlessly. The only drawback of the design is the extra traffic generated from synchronizing the MoM's and MLMs' databases. Fortunately, the traffic can be reduced by proper placement of MLMs or by using dedicated connections between pairs of MLMs.

The features of FTNMS can be summarized as:

- Reliable and fault-tolerant.
- Affordable.
- Easily implemented.
- Seamlessly incorporated with existing network management systems.
- Scalable to organization's needs.
- Can be implemented using different topologies.
- Requires no changes in the management protocol or the management application.
- A centralized and hierarchical network management system.
- Uses of-the-shelf components.
- Offers a centralized view/control of both system and network.

Chapter 9

Conclusion and Future Work

This chapter summarizes the contributions made in this thesis, future work is also proposed at the end of the chapter.

9.1 Conclusion

The thesis discussed fundamental aspects of network management and presented the reliability problem in such systems. In the thesis, we discussed a new framework for improving the reliability of network management systems and its design issues, implementing details, evaluation and comparison with similar solutions.

The following contributions were made in this thesis:

- Introducing a new paradigm that can improve the reliability of both centralized and hierarchical network management systems and may provide a new approach for a similar application in distributed network management systems.

- Providing an affordable, scalable, and flexible framework that can be easily incorporated into existing NMSs without the need for modifications neither in software nor in protocols.

- Providing a survey of fault-tolerance aspects of NMSs that presents the related work in the area of network management systems' reliability.

- Providing a new innovative pairing-based redundancy that can improve the reliability of a hierarchical NMS with the minimum cost.

- Applying the High-Availability technology concepts to existing network management systems.

The results show that the proposed system is easy to implement and that it can improve the reliability of existing network management systems. The only obvious cost will be in the increase of the traffic needed to synchronize between the nodes.

9.2 Future Work

The following issues might be considered in the future:

- Investigating the possibility of extending the design to distributed network management systems.

- Studying the possibility of implementing an active-active mode in the FTNMS-MoM level.

- Providing a more intensive study about the placement of nodes and its effect on the traffic generated.

- Checking for the possibility of race conditions where managers may be competing for the primary rule.

- Measuring the probability of having inconsistent databases at any point of time.

- Measuring the delay in reporting traps resulting from introducing the FTNMS-MLM layer.

- Performing real measurements of the traffic generated by the system on a real environment. The experiment includes setting up a reasonable number of agents or traffic generators to allow for an estimation of the overhead traffic under different operating conditions.

- Providing more accurate estimate of availability by changing the assumption that a failure of one pair of MLM causes the system to be unavailable. This assumption provides the worst case availability. However, the actual availability will be better if this assumption is relaxed.

Appendix A

Acronyms

ARP Address Resolution Protocol

CA Continuous Availability

CMIP Common Management Information Protocol

Disman Distribute Management

DRBD Distributed Replicated Block Device

FTNMS Fault-Tolerant Network Management System

GPL General Public License

HA High-Availability

HI-ADSD Hierarchical Adaptive Distributed System-level Diagnosis

HSRP Hot-Standby Router Protocol

HTTP Hypertext Transfer Protocol

IEEETFCC IEEE Task Force on Cluster Computing

IETF Internet Engineering Task Force

ISO International Standards Organization

IP Internet Protocol

IPC Inter-Process Communication

LVS Linux Virtual Server

MA Management Application

MAC Media Access Control

MD Management Distributor

MIB Management Information Base

MLM Middle Level Manager

MoM Manager of Managers

MTBF Mean Time Between Failure

MTTF Mean Time To Fail

MTTR Mean Time To Repair

NIC Network Interface Card

NMS Network Management Systems

RAID Redundant Array of Independent (or Inexpensive) Disks

RBD Reliability Block Diagram

Remop Remote Operation

RIP Real IP

RMON Remote Monitoring

RNMP Reliable Network Management Platform

RP Request Broker

SCSI Small Computer System Interface

SMI Structure of Management Information

SNMP Simple Network Management

SPOF Single Point Of Failure

STONITH Shoot The Other Node In The Head

TCP Transmission Control Protocol

UDP User Datagram Protocol

UML User-Mode Linux

VIP Virtual IP

VM Virtual Machine

Appendix B

System Snapshots

This appendix presents snapshots of some of the tools used in this work. Section B.1 shows few snapshots of the nPulse NMS application used with MoMs. Section B.2 however, shows the interface of the applet used to control MLMs.

B.1 nPulse

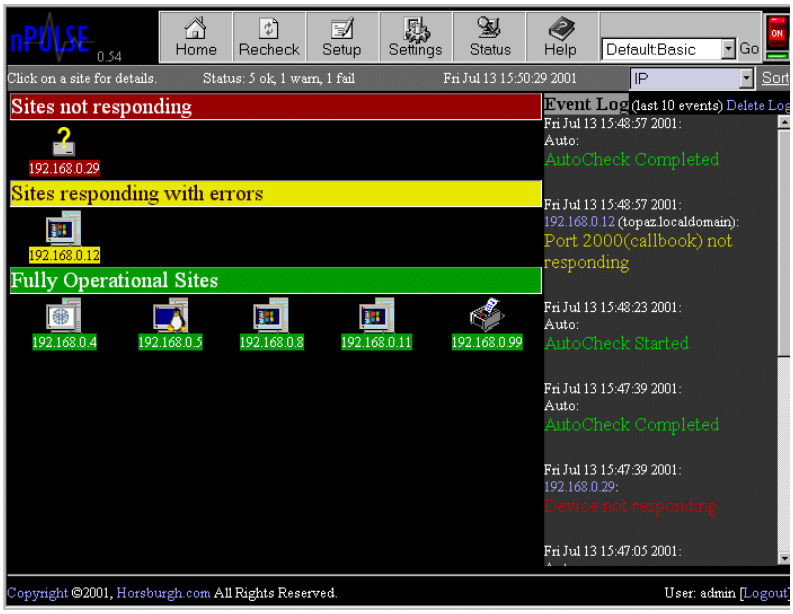


Figure B.1: nPulse: Overview Status Screen.

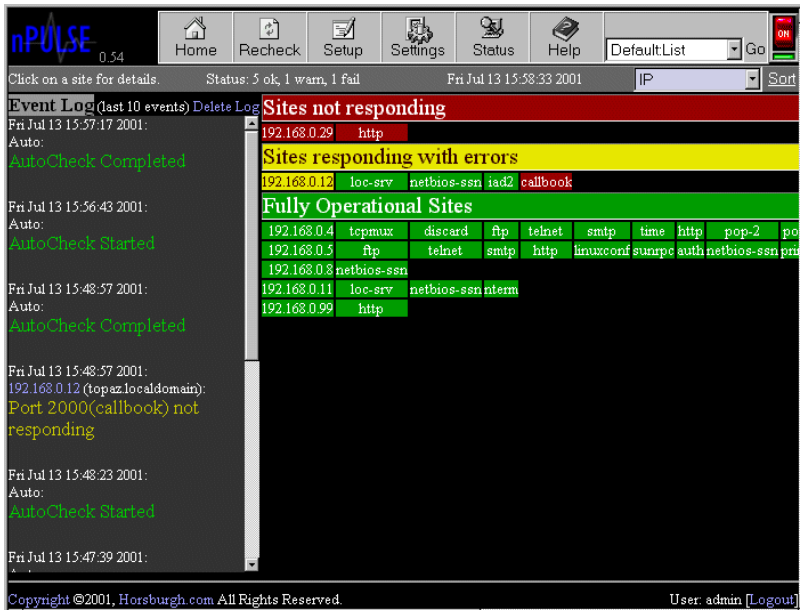


Figure B.2: nPulse: Listing Status Screen.

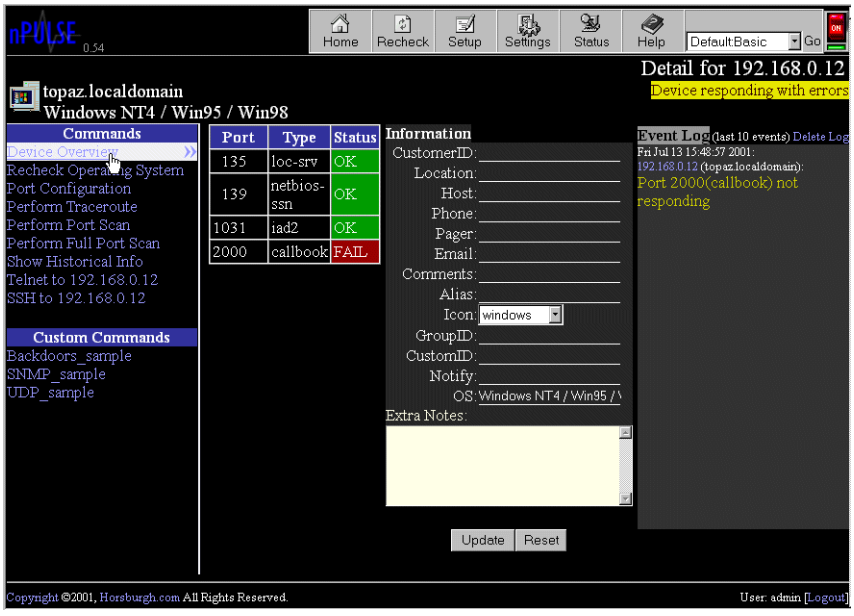


Figure B.3: nPulse: Detailed Device View.

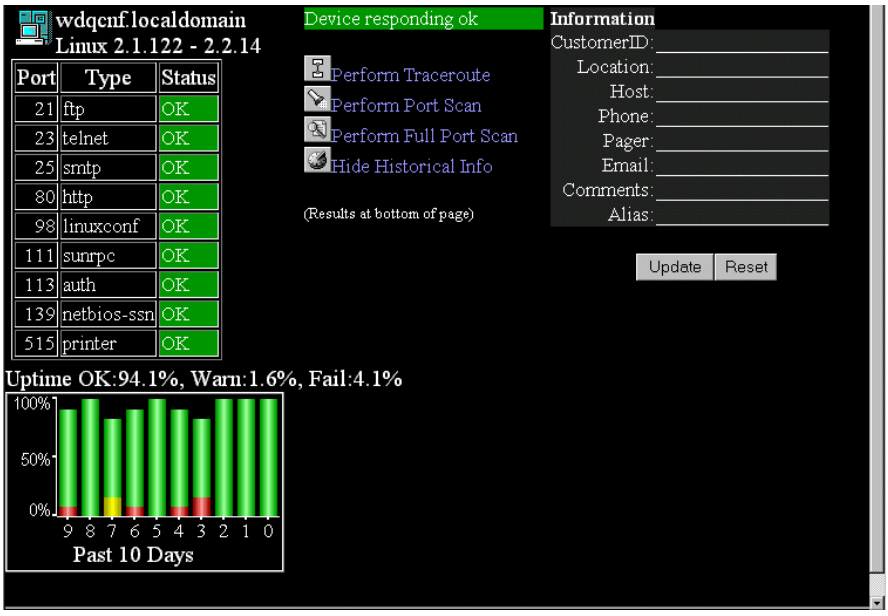


Figure B.4: nPulse: Historical Site Status.

B.2 FTNMS Applet

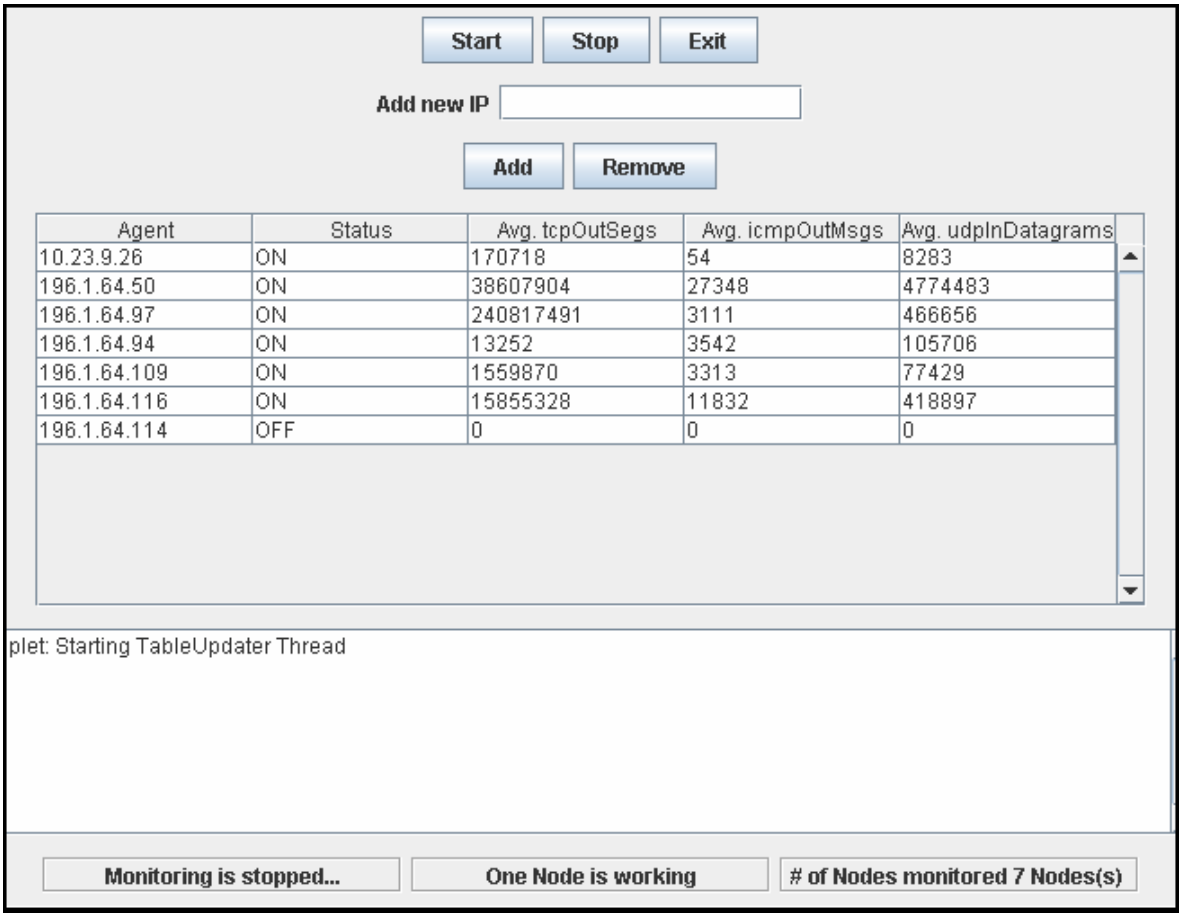


Figure B.5: FTNMS: Web-based Controlling Applet.

Appendix C

SHARPE & Matlab Codes

C.1 SHARPE Codes

C.1.1 SHARPE Code for Hierarchical-1 Markov Chain

```
format 8
factor on

markov Hierarchical2(lambda, mu)
S12 S02 lambda
S12 S11 2*lambda
S02 S12 mu
S02 S01 2*lambda
S11 S12 mu
S11 S01 lambda
S11 S10 lambda
S01 S02 mu
S01 S11 mu
```

```
S01 S00 lambda
S10 S11 2*mu
S10 S00 lambda
S00 S01 2*mu
S00 S10 mu
end
end

echo*****************************************************
echo ***** Outputs asked for the model: Hierarchical2*****

end
```

C.1.2 SHARPE Code for FTNMS-1 Markov Chain

```
format 8
factor on

markov FTNMS1(lambda, mu)
S22 S21 2*lambda
S22 S12 2*lambda
S21 S22 mu
S21 S11 2*lambda
S21 S20 lambda
S12 S22 mu
S12 S11 2*lambda
```

```
S12 S02 lambda
S11 S12 mu
S11 S21 mu
S11 S10 lambda
S11 S01 lambda
S02 S12 2*mu
S02 S01 2*lambda
S20 S21 2*mu
S20 S10 2*lambda
S10 S20 mu
S10 S11 2*mu
S10 S00 lambda
S01 S11 2*mu
S01 S02 mu
S01 S00 lambda
S00 S01 2*mu
S00 S10 2*mu
end
end

echo ****************************************************
echo *********  Outputs asked for the model: FTNMS1 *******

end
```

C.1.3 SHARPE Code for Hierarchical-4 Markov Chain

```
format 8
factor on

markov Hierachical4(lambda, mu)
S14 S04 lambda
S14 S13 4*lambda
S04 S14 mu
S04 S03 4*lambda
S13 S14 mu
S13 S03 lambda
S13 S12 3*lambda
S03 S13 mu
S03 S04 mu
S03 S02 3*lambda
S02 S12 mu
S02 S03 2*mu
S02 S01 2*lambda
S01 S02 3*mu
S01 S00 lambda
S01 S11 mu
S12 S13 2*mu
S12 S11 2*lambda
S12 S02 lambda
S11 S12 3*mu
```

```
S11 S10 lambda
S11 S01 lambda
S10 S11 4*mu
S10 S00 lambda
S00 S10 mu
S00 S01 mu
end
end

echo*****************************************************
echo*******  Outputs asked for the model: Hierachical4 ***

end
```

C.1.4 SHARPE Code for FTNMS-2 Markov Chain

```
format 8
factor on

markov FTNMS2(lambda, mu)
222 122 2*lambda
222 212 2*lambda
222 221 2*lambda
122 222 mu
122 022 lambda
```

```
122 112 2*lambda
122 121 2*lambda
212 222 mu
212 112 2*lambda
212 202 lambda
212 211 2*lambda
221 222 mu
221 121 2*lambda
221 211 2*lambda
221 220 lambda
022 122 2*mu
022 012 2*lambda
022 021 2*lambda
112 122 mu
112 212 mu
112 012 lambda
112 102 lambda
112 111 2*lambda
121 122 mu
121 221 mu
121 021 lambda
121 111 2*lambda
121 120 lambda
211 212 mu
211 221 mu
211 111 2*lambda
```

```
211 201 lambda
211 210 lambda
220 221 2*mu
220 120 2*lambda
220 210 2*lambda
202 212 2*mu
202 102 2*lambda
202 201 2*lambda
012 022 mu
012 112 2*mu
012 002 lambda
012 011 2*lambda
021 022 mu
021 121 2*mu
021 011 2*lambda
021 020 lambda
102 112 2*mu
102 202 mu
102 002 lambda
102 101 2*lambda
111 112 mu
111 121 mu
111 211 mu
111 011 lambda
111 101 lambda
111 110 lambda
```

```
120 121 mu
120 220 mu
120 020 lambda
120 110 2*lambda
201 202 mu
201 211 2*mu
201 101 2*lambda
201 200 lambda
210 211 2*mu
210 220 mu
210 110 2*lambda
210 200 lambda
002 012 2*mu
002 102 2*mu
002 001 2*lambda
011 012 mu
011 021 mu
011 111 2*mu
011 001 lambda
011 010 lambda
020 021 2*mu
020 120 mu
020 010 2*lambda
101 102 mu
101 111 2*mu
101 201 mu
```

```
101 001 lambda
101 100 lambda
110 111 2*mu
110 120 mu
110 210 mu
110 010 lambda
110 100 lambda
200 201 2*mu
200 210 2*mu
200 100 2*lambda
001 002 mu
001 011 2*mu
001 101 2*mu
001 000 lambda
010 011 2*mu
010 020 mu
010 110 2*mu
010 000 lambda
100 101 2*mu
100 110 2*mu
100 200 mu
100 000 lambda
000 001 2*mu
000 010 2*mu
000 100 2*mu
end
```

```
end

echo ****************************************************
echo *********  Outputs asked for the model: FTNMS2 *********

end
```

C.2 Matlab Codes

C.2.1 Matlab Code for Hierarchical-2 Availability

```
syms p12 p11 p02 p01 p10 p00 lambda mu;
eqn1='0= (-3* lambda)* p12+ mu* p11+ mu* p02';
eqn2='0= lambda* p12 - (2* lambda+ mu)* p02 + mu* p01';
eqn3='0= 2* lambda* p12- (2* lambda+ mu)* p11+ 2* mu* p10+ mu* p01';
eqn4='0= lambda* p11- (lambda+ 2* mu)* p10+ mu* p00';
eqn5='0= 2* lambda* p02+ lambda* p11- (lambda+ 2* mu)* p01+ 2*
mu* p00';
eqn6='0= lambda* p10+ lambda* p01- (3* mu)* p00 ';
eqn7='p12+p11+p02+p01+p10+p00=1';
A=solve(eqn1,eqn2,eqn3,eqn4,eqn5,eqn7,'p12','p11','p10','p02',
'p01','p00');
```

C.2.2 Matlab Code for FTNMS-1 Availability

```
syms p22 p12 p21 p11 p20 p02 p10 p01 p00 lambda mu;
eqn1='0 = -4 * lambda* p22 + mu * p12 + mu * p21';
eqn2='0 = 2 * lambda * p22 - (3* lambda + mu)* p12 + mu * p11 + 2*
 mu * p02';
eqn3='0 = 2 * lambda * p22 - (3* lambda + mu)* p21 + mu * p11 + 2*
 mu * p20';
eqn4='0 = 2 * lambda * p12 + 2 * lambda * p21 -(2* lambda +2 * mu)*
 p11+ 2* mu
 * p10 + 2 * mu * p01';
eqn5='0 = lambda * p12 - (2* lambda +2 * mu)* p02+ mu * p01';
eqn6='0 = lambda * p21 - (2* lambda +2 * mu)* p20+ mu * p10';
eqn7='0 = lambda * p11 + 2 * lambda * p20 - (lambda+3*mu)* p10 - 2 *
 mu * p00';
eqn8='0 = lambda * p11 + 2 * lambda * p02 - (lambda+3*mu)* p01 - 2 *
 mu * p00';
eqn9='0 = lambda * p10+ lambda * p01 - 4 * mu* p00';
eqn10='p22+p12+p21+p11+p20+p02+p10+p01+p00=1';
A=solve(eqn1,eqn2,eqn3,eqn4,eqn5,eqn6,eqn7,eqn8,eqn10,'p22','p12',
'p21','p11',
'p20','p02','p01','p10','p00');
```

Appendix D

Virtualization Using UML: Overview, Installation & Configuration

D.1 Virtualization Technology

A Virtualization environment will be used to test the proposed design. This will eliminate the need of multiple physical machines to test the proposed system. Because of that it is important to set the floor by presenting some background information on this topic. In this section we will present the concept of virtualization and discuss some of the virtualization software available. A major part of this section is devoted to User-Mode Linux (UML), a tool, that will be used for testing.

D.1.1 What is Virtualization?

Virtualization *is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others* [51]. This allows more than one virtual

machine (VM) to be running simultaneously under one physical machine. In this context, a virtual machine is an operating system environment that runs inside another operating system environment.

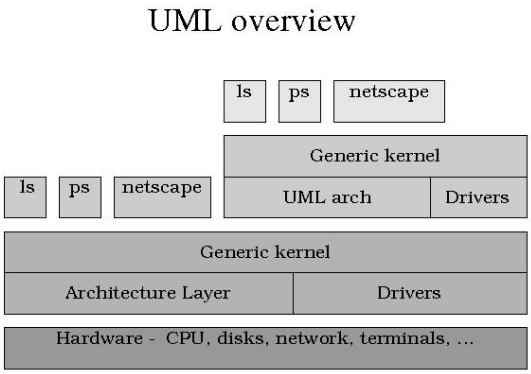


Figure D.1: UML as a System Process.

As in Figure D.1, virtualization applications run in the application level in top of an operating system. The general model consists of a host OS and a guest OS. *The host OS is the OS running on the physical machine and running the virtualization application. The guest OS on the other hand, is the OS running inside the virtual machine created by the virtualization application*. The guest OS runs independently from the host machine or other VMs running on the same host. Each VM is given its own dedicated set of resources (hard drive partitions, memory locations, virtual network interface, etc.) from the host [52].

As example of applications of virtualization, we can run different instances of different operating systems on one machine, simulating a virtual network with multiple nodes. Moreover, by configuring some nodes as software switches/routers, more complicated scenarios can be made.

D.1.2 Benefits of Virtualization

Virtualization found its way in many applications including the following:

- In cases of having under-utilized servers, servers can be grouped into one physical machines. This cuts down the costs needed for hardware, maintenance, adminstration and management.

- When requiring legacy applications that need a legacy hardware or old operating system.

- It also can be used when a user need to run untrusted application download from the internet for example.

- Using virtualization software we can give an illusion of hardware that we do not have to software that uses such hardware. For example, an additional NIC card or SCSI disk, etc.

- Virtualization can be also used when requiring more than one OS to run on the same machine simultaneously.

- It can be also used for software testing by providing safe environment instead of testing on a space that contains valuable data such as a desktop or a laptop computer.

- In teaching, virtualization can be used to teach students practical networking with out having many physical machines.

- Virtualization found it is way into virtual web hosting for users who want small web site but with its own IP address [53].

D.1.3 Some Virtualization Tools

There exist many virtualization software both commercial and non-commercial. Examples of commercial virtualization products are:

1. Microsoft Virtual PC (bought from Connectix) [54]
2. Microsoft Virtual Server [55]
3. VMware Workstation [56]
4. IBM z/VM [57]

On the other hand, non-commercial softwares include:

1. User-Mode Linux (UML) [58]
2. Linux VServer [59]

Different virtual machines vary in performance, supported operating systems, supported hardware, networking, ease of installation. For example, Microsoft Virtual PC 2004 and VMware Workstation 5 support Microsoft Windows, Mac OS, Linux, Novell NetWare and IBM OS/2 and others as guest OS [60][56]. On the other hand, UML and Vserver only support Linux as guest OS.

The tools also support different host OSs. For Instance, VMware Workstation can be installed on Windows, Linux and Mac OS systems. However, Microsoft Virtual PC only works with Windows and Mac OS. A comprehensive comparison between many virtual machines can be found in [61].

D.1.4 User Mode Linux (UML)

User Mode Linux (UML) is a Linux port that allows instances of Linux OS to run on inside another Linux OS. UML project has been started since 1999 by Jeff Dike. After that, many

people contributed to UML by making patches and fixing bugs.[58]

Compared to other virtualization tools, UML has many advantages including:

Supporting Linux: Since Linux is preferred over Windows for scientific research due to it is flexibility, open-source nature and availability of scientific tools. In addition, UML can be run on any Linux distribution (Host), and can run any Linux distribution (Guest).

Available for Free: Whereas commercial virtualization tools are expensive(129$ for Microsoft Virtual PC and 190$ for Vmware Workstation), UML is available for free.

Open Source: The open source nature of UML allows for development, since it is released under the GNU General Public License (GPL).

Flexibility: UML allows you to the virtual machines with the linux kernel of your choice and the specifications we need.

Speed: UML runs as a host process and not as a complete application. It also does not emulate any hardware layer like other applications. Hence, it allows us to install the minimum VM with a command prompt, this reduces significantly the overhead experienced by the GUI part of the OS and the overhead of the hardware emulation.

Fault-Tolerance Testing Add-ons The availability of fault tolerance testing tools such as FAUmachine [62] (previously known as UMLinux)for UML, makes it very suitable for our research work.

Maturity: UML started long time ago and has been tested by many people and for many applications. It was also reported to be used by some commercial companies to provide a virtual web hosting[53]. Support is also available through the websites and mailing lists.

Synchronization with Latest Linux Development: Since any Linux kernel source can be used to generate a UML binary, UML automatically includes the latest Linux features.

Attracting Features: UML has many attracting features like networking support, host file system access, compatibility with host binaries.

UML is easy to compile and install. During the installation, you can select and kernel version to your VM. You can also fully configure the kernel to include the features you want. Moreover, You can run any Linux distribution inside the VM by selecting the corresponding file system. Once installed, you can use it as an ordinary Linux machine, install applications, run network services, add users, etc. We are going to see this shortly in next section.

In case you want to upgrade the kernel or the file system, you can do each independently. I.e., you can upgrade the kernel without losing your data. Alternatively, you can change the file system with out changing the kernel. You can also have many UML instances sharing one file system. The later reduces the space requirements significantly.

D.2 UML Installation & Configuration

Installing UML is a simple but a tricky process. The process consists of two phases, a mandatory phases and an optional one:

1. Patching and compiling the UML Kernel. (Mandatory)
2. Patching and recompiling the host kernel.(Optional)

The results of the first phase is an executable UML binary. The second phase modifies the host's kernel to run virtual machines faster and to separate the host OS space from the UML space, hence hiding the UML processes from the host OS.

D.2.1 Patching and Compiling the UML Kernel

In this phase, two pieces of software are needed. The first is a vanilla source kernel, this can be downloaded from the www.kernel.org. In this step, you can download the kernel version

of your choice. The second is a UML patch that corresponds to the kernel source you downloaded earlier, the patch can be downloaded from http://user-mode-linux.sourceforge.net. More recent patches can be found in http://www.user-mode-linux.org/ blaisorblade/patches.

You need also to download (or make) a file system to attache to your virtual machine. The site user-mode-linux.sourceforge.net contains a variety of pre-built file systems corresponding to almost all linux distributions. You can either download a pre-built file system or build one using tools such as mkrootfs[63], UML Builder[64], gBootRoot[65] and rootstrap[66].

For this example, I will use the pre-build file system root_fs.rh-7.2-full.pristine.20020312.bz2 I downloaded from the user-mode-linux.sourceforge.net. From the name, we can infer that the file system corresponds to redhat 7.2.

For the purpose of this discussion, we will be using kernel version 2.6.8.1 as an example.

1. Download the kernel source tarball **linux-2.6.8.1.tar.bz2** or **linux-2.6.8.1.tar.gz** from http://www.kernel.org.

2. Download the corresponding **uml-patch** from user-mode-linux.sourceforge.net. For this kernel version we need uml-patch-2.6.8.1-1.bz2. Note, that that last digit 1 indicates the patch version. So, if more that one patch version exist, select the more recent one.

   ```
   NOTE: The availability of an exactly matching patch is not
   guaranteed.  Hence, if you do not find the right patch, look
   for a very close one.  This trick works most of the time.
   ```

3. Unpack the kernel source using one of the following:

   - If you downloaded the kernel in the *bz2* format

     tar -jvxf linux-2.6.8.1.tar.bz2

   - If you downloaded the kernel in the *gz* format

     tar -zvxf linux-2.6.8.1.tar.gz

This will take few minutes to complete depending on the speed of your machine. When done, a new folder will be created called linux-2.6.8.1 containing the uncompressed kernel source. If you download your kernel source from unofficial site, or if you are not sure that the source kernel is clean (meaning it has been configured or compiled before), you can use the following to have a clean kernel source

make mrproper

4. Move the **uml-patch** to the **linux-2.6.8.1** directory

   mv uml-patch-2.6.8.1-1.bz2 linux-2.6.8.1

5. Unpack and apply the patch

   bzcat uml-patch-2.6.8.1-1.bz2 | patch -p1

   If you want to apply any other patches, apply them now. But be careful, some patches might conflict with each other. If you got an error in this step, it means that either the patch or the kernel source is wrong.

6. Configure the kernel

   make menuconfig ARCH=um

   The ARCH=um parameter is crucial because it tells the kernel build process to build a UML kernel rather than a native x86 kernel. In this step, select the features you would like to include in your kernel virtual machine.You can think of this as selecting the features supported by the guest OS.

   To provide the UML instance with networking connectivity, you must enable the following options:

   Network device support $\Rightarrow$ Universal TUN/TAP device driver support

   This option is selected because we will be using networking with bridging and tunneling. It is worth to mention that networking in UML can be provided in many ways

other than bridging. For more information, interested readers can visit [67].

7. Check the correct dependency between selected features.

   make dep # not necessary with kernel 2.6 or above

8. Compile the kernel

   make linux ARCH=um

   This the longest part of the process, it might take around half an hour on slow machines. The result of the compilation is a binary file called ”linux”, this is the file executed to run the UML.

9. If you selected any feature during the configuration as a module, you need to compile it now

   make modules ARCH=um

10. To install modules, you need first to mount your file system on the host machine. Before mounting, we need to unpack the file system if its packed.

    tar -jvxf root_fs.rh-7.2-full.pristine.20020312.bz2

    ```
    In this example, I am assuming that the root files system is
    in the root directory of the kernel source.  If it is not the
    case with you, then either move it to the linux-2.6.8.1 directory
    or use the exact path in all coming commands.
    ```

    For simplifying the the running process, rename the file system to root_fs as it is the default name the ”linux” is going to look for. Now, mount the files system to the directory /mnt/mnt

    mount root_fs /mnt/mnt -o loop

    then install the modules

    make modules_install INSTALL_MOD_PATH=‘pwd‘/mnt/mnt ARCH=um

11. Optionally, you can reduced the size of the ”linux” binary using

    strip linux

    This is due to the fact that the ”linux” binary is made mostly of symbols that are not needed. This **strip** command removes those symbols and saves some space.

12. To run UML

    ./linux ubd0=root_fs MEM=32MB

    Where:

    ubd0 : Is the mount point of the root file system associated with the UML instance.

    MEM: The amount of memory assigned to the UML instance.

    Other options also exist, we will see some when making the networking setup.

    A boot screen is going to appear and a login prompted will show. Most file systems have the following default login

    user: root

    password: user

    Only in the first login, you will be sked to change your password. Once logged on, you can enjoy Linux/UNIX commands such as ls, cd, ps, etc.

D.2.2 Setting Up The UML for Networking

As it has been compiled, our UML does not need further work to provide networking, except for setting IP address, DNS, Gateway, etc. The remaining part is going to be on the host machine. As mentioned earlier, UML can be set up for networking in many ways. But for our work, I am going to show how to provide networking using bridging[68].

In order to provide our UML instance with a network connectivity, we need to install two packages : *bridge-utils* and *uml_utilities* on the host machine. The packages includes tools to

create bridges and tunnels needed to establish a virtual network.

1. The bridge-utils package can be downloaded in RPM format from http://bridge.sourceforge.net/.

2. To install bridge-utils package type.

   rpm -ivh bridge-utils-0.9.6-1.i386.rpm

   You can test the package by running

   brctl

   to see if the command is defined

3. The uml_utilities can be downloaded from http://jungla.dit.upm.es/ vnuml/download/rpms/.

   You can test the package by running

   tunctl

   to see if the command is defined

4. To install the uml_utilities package, type

   rpm -ivh uml_utilities-20021103-1.i386.rpm

5. Logging as a root, you need to change the access rules of the following directory.

   chmod 660 /dev/net/tun

   For the access rules, you can either allow only the root to have read/write permission or make a new group and give the users within the group a read/wirte permission. If only the root have a permission, then he will be the only one who can run UML with networking.

6. Setup eth0 to have IP=0.0.0.0 and be promiscuous.

   ifconfig eth0 0.0.0.0 promisc up

7. Create a new bridge called *uml-bridge* with nothing connected to it.

   brctl addbr uml-bridge

8. Set the forwarding delay, to zero seconds

   brctl setfd uml-bridge 0

9. Since we have only one bridge, we will set the Spanning Tree Protocol ”hello” packet to 0, meaning off.

   brctl sethello uml-bridge 0

   brctl stp uml-bridge off

10. Assign the IP address of the physical machine to the bridge.

    ifconfig uml-bridge 172.16.70.132 netmask 255.255.0.0 up

11. Add the physical interface eth0 to the uml-bridge

    brctl addif uml-bridge eth0

12. For each interface on each UML instance create a tunnel and add it to the bridge.

    tunctl -u root -t uml-conn0

    ifconfig uml-conn0 0.0.0.0 promisc up

    brctl addif uml-bridge uml-conn0

    the ”root” in the first command refers to the owner of the tunnel. If it is not the root, then put the right owner. For example, a uml-users group you might have defined in step 5 above.

13. Now run UML, the command line must look like this

    ./linux ubd0=root_fs MEM=32MB eth0=tuntap,uml-conn0,FE:FD:0:0:0:A

    Tuntap refers to the tunneling mechanism we are using for networking. *Uml-conn* is the tunnel name and you need to assign an arbitrary MAC address. If you have more than one NIC interface on the same UML, you need to have a different tunnel for each and to include it in the command line.

14. Once logged on, make the necessary network setting such as DNS, default Gateway, IP address, etc. Finally, try to ping to and from the UML to test the connectivity.

15. After finishing using UML, you may want to go back to your old network setting. To do this you need to shutdown the bridges and tunnels you made and assign eth0 its old IP. Here are the steps:

    shutdown the uml-bridge

    ifconfig uml-bridge down

    remove the interfaces from the bridge

    brctl delif uml-bridge eth0

    brctl delif uml-bridge uml-conn0

    Delete the uml-bridge

    brctl delbr uml-bridge

    Delete the all the tunnels

    tunctl -d uml-conn0

    Bring eth0 back with its old IP

    ifconfig eth0 172.16.70.132 netmask 255.255.0.0 up

    Assign the default gateway again if necessary

    route add default gw 172.16.0.253

D.2.3 Patching and Compiling the Host Kernel

The following steps are not mandatory for the basic operation of the UML. But, they will improve both the speed and the security of the machine on which your are running UML. In this procedure, a patch called Seperate Kernel Adress Space (SKAS) is applied to the host kernel. After that, the host kernel is recompiled.

The philosophy behind the SKAS patch is that UML works basically in what is called a Tracing Thread (tt) mode. In the tt mode, the kernel exists in the same address space as its process, which obviously makes it writable and hence imposes some security risks. In this mode, the UML also uses a signals to force control of the UML system calls or control. The signal delivery and return are slow and degrades the performance significantly.

To overcome those problems, a SKAS mode is used. In this mode, the kernel exists in different address space than its processes and hence the name comes. The SKAS mode also, eliminates the need for the system signals delivery and hence improves the performance.

Before we start the procedure, it would be good to mention some notes about the kernel recompilation process. Kernel recompilation involves configuring the kernel with many specifications including hardware and software parameters that even advanced users might not be familiar with. The configurations are saved into a file in the kernel source directory that is called ".config". Because of the risk involved with this configuration, it is much easier to find a ".config" file that has been already configured against your system.

For our example, I will show how to extract the ".config" file of Linux Fedora Core 3 with Linux Kernel 2.6.9. In this example, my host kernel is 2.6.9 and I am going to upgrade to version 2.6.10 along with the SKAS application process.

1. Download the source RPM for the host kernel from http://download.fedora.redhat.com/pub/fedora/linux/core/3/i386/os/SRPMS/. For this example, I will be using kernel-2.6.9-1.667.src.rpm.

2. Install the kernel

   rpm -ivh kernel-2.6.9-1.667.src.rpm

   rpmbuild −bp −−target=noarch /usr/src/redhat/SPECS/kernel-2.6.spec

3. Check the architecture of your machine

   uname -m

The output will be i686 or i386, etc. For my case, I got i686. Hence, the ".config" file will be located in /usr/src/redhat/BUILD/kernel-2.6.9/linux-2.6.9/configs/kernel-2.6.9-**i686**.config.

Now the configuration file is ready. We will use it with our new kernel in a while.

Now let's download the new kernel (2.6.10) and apply the SKAS patch to it:

1. Download a new kernel from http://www.kernel.org. The corresponding source tarball will be linux-2.6.10.tar.bz2. Put the downloaded tarrball into /usr/src/ directory.

2. Untar the kernel source

   tar -jvxf linux-2.6.10.tar.bz2

   A directory called "linux-2.6.10" will be generated.

3. Download the SKAS patch from http://user-mode-linux.sourceforge.net. The most recent patches can be found in http://www.user-mode-linux.org/ blaisorblade/patches. The corresponding SKAS patch is skas-2.6.10-v8.patch.bz2.

4. Untar the SKAS patch

   bzip2 skas-2.6.10-v8.patch.bz2

5. Apply the SKAS patch

   patch -p1 -d linux-2.6.10 $\prec$ skas-2.6.10-v8.patch

6. To make sure the the source kernel is clean, cd into the kernel directory and

   cd linux-2.6.10 make mrproper

7. Copy the ".config" file into the kernel source directory.

   cp /usr/src/redhat/BUILD/kernel-2.6.9/linux-2.6.9/configs/kernel-2.6.9-i686.config .config

8. To copy the configurations from the existing installation

   make oldconfig

   During this step, if you come across an option that you do not know, just press **Enter** to select the default.

9. Configure the kernel with the options of your choice

   make menuconfig

   For UML, you need to enable the **Processor Types & Features** $\Longrightarrow$ **"/proc/mm"**.

10. Compile the kernel using

    make all

11. Finally, compile and install modules

    make modules_install

    make install

    When you reboot, you will find two options in the boot screen, the old one and the new one with skas suffix. The later is the one you will use to run UML.

Appendix E

Linux HA Installation & Configuration

E.1 Heartbeat Installation

To install the heartbeat package, you need to download the some packages from http://www.ultramonkey.org. In this example we are going to use UltraMonkey package version 2.0.2. The following packages must be downloaded:

heartbeat-2.0.2-1.i386.rpm
heartbeat-ldirectord-2.0.2-1.i386.rpm
heartbeat-pils-2.0.2-1.i386.rpm
heartbeat-stonith-2.0.2-1.i386.rpm

In addition, some dependency packages must also be installed before installing the heartbeat packages if they do not exist. Here are some of them:

libnet-1.1.2.1-1.rh.el.um.1.i386.rpm
perl-Authen-SASL-2.08-1.rh.el.um.1.noarch.rpm
perl-Convert-ASN1-0.18-1.rh.el.um.1.noarch.rpm
perl-Crypt-SSLeay-0.51-4.rh.el.um.1.i386.rpm

perl-IO-Socket-SSL-0.96-1.rh.el.um.1.noarch.rpm

perl-Mail-IMAPClient-2.2.9-1.rh.el.um.1.noarch.rpm

perl-Parse-RecDescent-1.94-1.rh.el.um.1.noarch.rpm

perl-XML-NamespaceSupport-1.08-1.rh.el.um.1.noarch.rpm

perl-XML-SAX-0.12-1.rh.el.um.1.noarch.rpm

perl-ldap-0.3202-1.rh.el.um.1.noarch.rpm

A simple instruction to install all packages is

rpm -ivh heartbeat*

After finishing the installation, a folder called /etc/ha.d that contains the configuration files for Heartbeat.

E.1.1 Heartbeat Configuration

Before starting using Heartbeat, four files need to be configured:

1. /etc/ha.d/ha.cf - which controls the heartbeat details.
2. /etc/ha.d/haresources - which defines the VIP and the resources to acquire when the primary node fails
3. /etc/ha.d/conf/ldirectord.cf - Which defines the real servers, services to monitor, scheduling algorithms.
4. /etc/ha.d/authkeys

For the sake of illustration, In this section I will show and example of each configuration file. The example configuration assume two LDs using an ethernet heartbeat channel and a pool of two real servers providing http service.

ha.cf File The ha.cf file specify the parameters necessary for the heartbeat operation, such as the how long to wait before a host is declared as dead, the UDP port to send the heartbeats, etc. Below a sample ha.cf with minimum configuration is given:

```
debugfile /var/log/ha-debug
logfile /var/log/ha-log
logfacility local0
keepalive 2
deadtime 30
udpport 694
bcast eth0
auto_failback on
node lvs-ld1
node lvs-ld2
```

haresources The haresources file specify the resources to be claimed by the backup node when the primary node fails. Such resources include the VIP and the httpd daemon. The file also specify the primary node in case the auto fall back is on. Below a sample haresources with minimum configuration is given:

```
lvs-ld1 172.16.70.140 httpd
```

ldirectord.cf The ldirectord.cf is used to specify the real server cluster and some parameters for dealing with them. Similar to ha.cf, a set of global directives must be specified at the beginning of the file, then a group of directives is specified for each service provided by the cluster.

```
# Global Directives
checktimeout=3
checkinterval=1
fallback=127.0.0.1:80
autoreload=yes
logfile=”/var/log/ldirectord.log”
logfile=”local0”
quiescent=yes

# A sample virual with a fallback that will override the gobal
setting virtual=172.16.70.140:http
real=172.16.70.151:http masq
real=172.16.70.152:http masq
fallback=127.0.0.1:80 masq
service=http
request=”index.html”
receive=”That is”
virtualhost=lvs
scheduler=rr
protocol=tcp
```

authkeys The authkeys file is used to secure the heartbeats channel between the two or more involved nodes. Three security schemes can be used: Cyclic Redundancy Check (CRC), SHA Encryption and MD5 Encryption. CRC does not provide any security, but it is used against packet corruption. SHA is believed to be the best, and MD5 is the next best. Both SHA and MD5 require a key that must be the same between all nodes. The authkeys file permissions must be set to 600 so that it can only be read or written into by the system administrator.

```
auth 2
2 sha cluster1
```

E.2 DRBD Installation

DRBD can be installed in one of two ways: either as a part of the kernel or as a module. I am going to show how to install DRBD as a part of the kernel since its easier and applicable to

UML. For information about installing DRBD as a module, refer to http://www.drbd.org or http://www.linux-ha.org.

To install DRBD as a part of the kernel, a patch has to be compiled, then used to patch the kernel. Then, DRBD tools are compiled and installed. Let's start:

1. Download the DRBD package drbd-7.14.tar.gz from http://www.drbd.org.

2. Decompress the package

   tar -xzvf drbd-7.14.tar.gz

3. Change the directory to DRBD source and compile the patch. You must specify the kernel source of the kernel you are to patch

   cd drbd-7.14

   make KDIR=/usr/src/linux-2.6.8 kernel-patch

4. When the patch is compiled, use it to patch the kernel. For information, refer to "Patching and Compiling the host Kernel" section of UML installation.

5. When configuring the new kernel, make sure to select the DRBD under "Block Devices" options.

E.3 DRBD Configuration

All the configuration needed for DRBD to work are found in the file /etc/drbd.conf.

Where the "device" assigns the drbd device, there can be up to 8 drbd devices. "disk" defines the physical partition used by the drbd device. Before Using DRBD, you have to prepare a partition on each device to be used by DRBD. The "address" defines the IP and the port used by the DRBD daemon. Finally, the "meta-disk" defines where DRBD will keep information about changes that are made to the primary desk. Many other options can be added to the DRBD configuration file, for more information, visit http://www.drbd.org.

```
resource drbd0 {
protocol C;
incon-degr-cmd ”halt -f”;
on lvs-nms1 {
device /dev/drbd0;
disk /dev/VolGroup00/lvol0
address 172.16.70.131:7789;
meta-disk internal;
}
on lvs-nms2 {
device /dev/drbd0;
disk /dev/VolGroup00/lvol0;
address 172.16.70.132:7789;
meta-disk internal;
} }
```

Figure E.1: Example of a drbd.conf Configuration File.

After preparing the partitions and the configuration file, you can start the drbd service:

service drbd start

when you run DRBD for the first time, you must assign the primary role to one of the disk to force the two disks to synchronize:

drbdsetup /dev/drbd0 primary –do-what-I-say

to view the status of the DRBD disk you can execute

cat /proc/drbd

E.3.1 Integrating DRBD with Heartbeat

It is important to integrate DRBD with the heartbeat package. Otherwise, DRBD will not be of use. The integration is simple and is done using the /etc/ha.d/haresources file.

As we mentioned before, the ”haresources” file defines the resources shared by the two systems running the heartbeat. The resources are shutdown on the system that fails and ran on the system which takes over. The following entries can be added to the ”haresources” file

to integrate DRBD with Heartbeat.

drbddisk::drbd0 Filesystem::/dev/drbd/0::/mnt/drbd::ext2

Where the ”drbddisk” directive defines the drbd disk to be used and the ”Filesystem” defines the mounting directories and the filesystem type.Hence, a complete resources line can be

lvs-nms1 172.16.70.140 httpd snmpd drbddisk::drbd0
Filesystem::/dev/drbd/0::/mnt/drbd::ext2

E.3.2 Using DRBD to Synchronize Applications’ Data

You might be wondering now, how can DRBD be used to synchronize data used be a web server for example. The answer is simple and requires four steps:

- First, Create a mount point for the DRBD disk
  mkdir /mnt/drbd
- Second, mount the DRBD disk if not mounted on the mount point you created. Notice that the DRBD disk can only be mounted on the primary node.
  mount /dev/drbd0 /mnt/drbd
- Third, move the folder containing the web server data into the DRBD disk **mv /var/www/* /mnt/drbd/www**
- Finally, create a soft link with the same name as the folder you moved pointing to the folder on the DRBD disk **ln -s /mnt/drbd/www /var/www**

The link must be created on both nodes.If the application you have uses more than one folder, you might need to repeat the last two steps for each folder.

Appendix F

Configuration Files

F.1 MoM Configuration Files

Script to run MoM UML

./linux ubd0=mom1_root_fs

ubd1=swap-64-1 ubd2=drbd.ext2-256-1 eth0=tuntap,uml-mom1,FE:0:0:0:0:1

/etc/ha.d/ha.cf

 # File to write debug messages to

debugfile /var/log/ha-debug

 # File to write other messages to

logfile /var/log/ha-log

 # Facility to use for syslog()/logger

logfacility local0

keepalive 2

deadtime: how long-to-declare-host-dead?

deadtime 10 #warntime 10

#initdead 120

What UDP port to use for bcast/ucast communication?

udpport 694

#udp eth1

#bcast eth0 # Linux

ucast eth0 172.16.70.142

auto_failback on

#watchdog /dev/watchdog

node ftnms-mom1

node ftnms-mom2

/etc/ha.d/haresources

ftnms-mom1 172.16.70.150 httpd snmpd drbddisk::drbd0

Filesystem::/dev/drbd/0::/mnt/drbd::ext2 nPulse

/etc/ha.d/authkeys

auth 2

#1 crc

2 sha1 MoM12

#3 md5 Hello!

/etc/drbd.cf

resource drbd0 {

protocol C;

incon-degr-cmd ”halt -f”;

```
    on ftnms-mom1 {
device /dev/drbd/0;
disk /dev/ubd/disc0/disc;
address 172.16.70.151:7789;
meta-disk internal;
}
    on ftnms-mom2 {
device /dev/drbd/0;
disk /dev/ubd/disc0/disc;
address 172.16.70.152:7789;
meta-disk internal;
}
}
```

/etc/ha.d/resource.d/nPulse

```
#!/bin/sh
    ARGS="$0 $*"
    # Source function library.
prefix=/usr
exec_prefix=/usr
. /etc/ha.d/shellfuncs
    us='uname -n'
    usage() {
echo "./nPulse {start|stop|status}"
exit 1
}
```

```
    StartnPulse (){
/usr/local/npulse/etc/./start
echo ”START”
}
    StopnPulse () {
/usr/local/npulse/etc/./stop
echo ”STOP”
}
    # # See how we were called.
# # The order in which heartbeat provides arguments to resource
# scripts is broken. It should be fixed.
    case ”$#” in
0) echo ”At least 1 option must be given!”
usage
;;
    1)
cmd=$1
;;
    *)
echo ”Too many arguments”
usage;;
esac
    case ”$cmd” in
start) StartnPulse
;;
stop) StopnPulse
```

;;

Not quite sure what to do with this one...

We aren’t a continuously running service - so it’s not clear

#

status) echo ”stopped”

;;

*)

usage

;;

esac

exit 0

F.2 MLM Configuration Files

Script to run MLM UML

./linux ubd0=MLM1_root_fs ubd1=swap-64-1 ubd2=drbd.ext2-256-1

ubd3=drbd.ext2-256-2 eth0=tuntap,uml-mlm1,FE:0:0:0:0:1

/etc/ha.d/ha.cf

 # File to write debug messages to

debugfile /var/log/ha-debug

 # File to write other messages to

logfile /var/log/ha-log

 # Facility to use for syslog()/logger

logfacility local0

keepalive 2

deadtime: how long-to-declare-host-dead?

deadtime 10 #warntime 10

#initdead 120

What UDP port to use for bcast/ucast communication?

udpport 694

#udp eth1

#bcast eth0 # Linux

ucast eth0 172.16.70.152

auto_failback on

#watchdog /dev/watchdog

node ftnms-mlm1

node ftnms-mlm2

/etc/ha.d/haresources

ftnms-mlm1 drbddisk::drbd0 Filesystem::/dev/drbd/0::/mnt/drbd0::ext2 172.16.70.143

ftnms-mlm2 drbddisk::drbd1 Filesystem::/dev/drbd/1::/mnt/drbd1::ext2 172.16.70.144

/etc/ha.d/authkeys

auth 2

#1 crc

2 sha1 MLM12

#3 md5 Hello!

/etc/drbd.cf

```
resource drbd0 {
protocol C;
incon-degr-cmd ”halt -f”;
    on ftnms-mlm1 {
device /dev/drbd/0;
disk /dev/ubd/disc2/disc;
address 172.16.70.141:7789;
meta-disk internal;
}
    on ftnms-mlm2 {
device /dev/drbd/0;
disk /dev/ubd/disc2/disc;
address 172.16.70.142:7789;
meta-disk internal;
}
}
    resource drbd1 {
protocol C;
incon-degr-cmd ”halt -f”;

    on ftnms-mlm1 {
device /dev/drbd/1;
disk /dev/ubd/disc3/disc;
address 172.16.70.141:7788;
meta-disk internal;
```

```
}
    on ftnms-mlm2 {
device /dev/drbd/1;
disk /dev/ubd/disc3/disc;
address 172.16.70.142:7788;
meta-disk internal;
}
}
```

Appendix G

Troubleshooting

This appendix presents a troubleshooting references for some of the problems that may face one who wants to use some of the tools used in this work.

1. **UML instances can ping to any node on the network, but not to each other**
   This usually happens if both UML's have the same MAC address. Assign different MAC addresses to each UML when running the UML as follows:
   ./linux ubd0=root_fs eth0=tuntap,tun1,FE:FD:0:0:0:1
   do not rely on the default MAC assigned by UML, it might be the same for all virtual machines.

2. **When shutting down UML, it hangs at "Turning off swap"**
   Do the following
   cp -p /bin/true /sbin/hwclock

3. **UML provides 3 consoles for each virtual machine, can I make it one per machine?**
   Yes, you can remove the extra consoles by commenting the corresponding tty line in the /etc/inittab file. For example, you can leave tty1 uncommented and comment other tty lines.

4. **IPVSADM does not forward request to real server.**

   make sure that /proc/sys/net/ipv4/ip_forward is set to 1.

   cat /proc/sys/net/ipv4/ip_forward

   if it is not set to 1, set it using

   echo 1 > /proc/sys/net/ipv4/ip_forward

5. **Heartbeat service does not start**

   make sure that the permissions of the /etc/ha.d/authkeys file is set to 600

   chmod 600 /etc/ha.d/authkeys

6. **UML start killing httpd and other services**

   This usually caused by memory shortage. To solve this, add a swap space to your UML. Here the steps:

   - Download the swap file of the size you want from http://www.stearns.org/mkrootfs/filelist.html

   - Expand the file system

   cp -p –sparse=always ¡(cat emptyfs.swap.64.bz2 — bunzip2 -c -) swap

   chmod 644 swap

   Finally, instruct UML to use the swap space you made

   ./linux ubd0=root_fs ubd1=swap eth0=tuntap,tunn1,FE:FD:0:0:0:1

7. **How to partition a Logical Volume Manager(LVM) volume**

   - You have first to login into rescue mode. To do this, insert your linux first installation CD and select ”rescue mode”.

   - Unmount the following directories

   Umount /mnt/sysimage/dev

   Umount /mnt/sysimage/boot

   Umount /mnt/sysimage/sys

Umount /mnt/sysimage/proc

Umount /mnt/sysimage

- Check your file system for errors

E2fsck f /dev/VolGroup00/LogVol00

- Reduce the size of your file systems

Resize2fs /dev/VolGroup00/LogVol00 9G

-Enter to LVM console

lvm

- Reduce the size of the LVM volume

Lvreduce L 10G /dev/VolGroup00/LogVol00

-Create new LVM volume with a new size

Lvcreate L 1G VolGroup00

- Resize your old partition to occupy the left space of the volume

Resize2fs /dev/VolGroup00/LogVol00

Bibliography

[1] Mani Subramanian. *Network Management: Principles and Practice*. Addison Wesley, 2000.

[2] Joey De Wiele and Sameh Rabie. Meeting Network Management Challenges: Customization, Integration and Scalability. *Technical Program, Conference Record, IEEE International Conference on Communications, Geneva*, 2:1197–1204, May 1993.

[3] Nova Stars Networking Services. www.novastars.com/network-tutorials/lan-glossary.htm.

[4] E.P. Durate Jr. and Luis Carlos Erpen De Bona. A Dependable SNMP-Based Tool for Distributed Network Management. In *the Proceedings of the International Conference on Dependable Systems and Networks(DSN'02)*, pages 279–284, 2002.

[5] Tomohiro Fujisaki, Masaki Hamada, and Katsuyoshi Kageyama. A Scalable Fault-Tolerant Network Management System Built Using Distributed Object Technology. In *Proceedings of the 1st International Enterprise Distributed Object Computing Conference (EDOC '97), Gold Coast, Australia*, pages 140–148, October 1997.

[6] Soomyung Park, Sangjoon Ahn, Sunyoung Han, Wooyong Han, and Jungchan Na. The Reliable Distributed Network Management Platform. In *Proceedings of the 5th IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 439–445, 1995.

[7] Akhil Sahai and Christine Morin. Towards Distributed and Dynamic Network Management. In *the proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS), New Orleans, USA*, pages 455–464, Feb. 1998.

[8] Jean-Philippe Martin-Flatin, Simon Znaty, and Jean-Pierre Hubaux. A Survey of Distributed Enterprise Network and Systems Management Paradigms. *Journal of Network and Systems Management*, 7(1):9–26, March 1999.

[9] J.P. Martin-Flatin and S. Znaty. A Simple Typology of Distributed Network Management Paradigms. In *the Proceedings of the 8th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'97), Sydney, Australia*, pages 13–24, October 1997.

[10] F. StrauB. Distribution Models for Network Management Functions. Technical report, Technical University Braunschweig, http://www.ibr.cs.tu-bs.de/projects/jasmin/dismod.ps, 2000.

[11] F. Stamatelopoulos and N. Roussopoulos. Using a DBMS for Hierarchical Network Management. *Engineer Conference, NETWORLD+INTEROP'95 '95)*, March 1995.

[12] Rajesh Subramanyan, Jose Miguel-Alonso, and Jose A. B. Fortes. A Scalable SNMP-based Distributed Monitoring System for Heterogeneous Network Computing. In *the Proceedings of High Performance Networking and Computing Conference*, pages 52–52, 2000.

[13] S. Waldbusser. Remote Network Monitoring Management Information Base. Network Working Group, May 2000.

[14] T. Magedanz and T. Eckardt. Mobile Software Agents: a New Paradigm for Telecommunications Management. *Proceedings of the IEEE Network Operations and Management Symposium (NOMS96), Kyoto, Japan*, 2:360–369, April 1996.

[15] G. Goldszmidt, S. Yemini, and Y. Yemini. Network Management by Delegation - the MAD Approach. *Proceedings of the 1991 CAS Conference*, pages 347–359, 1991.

[16] OMG. The Common Object Request Broker: Architecture and Specification Revision 2.0, 1995.

[17] J. Siegel. *CORBA Fundamentals and Programming*. Wiley, New York, USA, 1996.

[18] F. Somers. HYBRID: Unifying Centralised and Distributed Network Management Using Intelligent Agents. *Proceedings of the IEEE Network Operations and Management Symposium (NOMS96),Kyoto, Japan*, 1:34–43, April 1996.

[19] Stephen S. Liu Thomas M. Chen. A Model and Evaluation of Distributed Network Management Approaches. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 20(4):850–857, May 2002.

[20] http://www.ietf.org/internet-drafts/draft-ietf-disman-remops-mib-v2 09.txt. ”definitions of managed objects for remote ping, traceroute, and lookup operations”.

[21] R. Boutaba and J. Xiao. Network Management: State of the Art. *Proceedings of IFIP World Computer Congress (WCC’02) TC6 Stream on Communication Systems: The State of the Art*, pages 127–146, 2002.

[22] Peter S. Weygant. *Clusters for High Availability: A Primer of HP Solutions*. Prentice Hall PTR, 2001.

[23] Linux High-Avalability Project Website. http://www.linux-ha.org.

[24] Linux Virtual Server Project Website. http://www.linuxvirtualserver.org.

[25] ”The Horus Project ”. http://www.thehorusproject.com/.

[26] ”The ISIS Project”. http://www.cs.cornell.edu/info/projects/isis/.

[27] ”IEEE Computer Society Task Force on Cluster Computing”. http://www.ieeetfcc.org/.

[28] Tau Leng, Christopher Stanton, and Shukri Zaibak. Architecting Linux High-Availability Clusters Part 2. *Dell Power Solutions*, 2:76–82, 2001.

[29] Sean Whalen. An Introduction to ARP Spoofing. Online Document, http://node99.org/projects/arpspoof, April 2001.

[30] Myung-Sup Kim, Mi-Joung Choi, and James W. Hong. A Load Cluster Management System Using SNMP and Web. *International Journal of Network Manangement*, 12(6):367–378, May 2002.

[31] Zhang W. Linux Virtual Server for Scalable Network Services. In *Ottawa Linux Symposium 2000*, July 2000.

[32] Kurose J. and Keith Ross. *Computer Networking A Top-Down Approach Featuring the Internet*. Addison Wesley, 2nd edition, 2003.

[33] Red Hat Website. http://www.redhat.com/products/software/linux/haserver/details.html.

[34] A. L. Robertson. The Evolution of the Linux-HA Project. *UKUUG LISA/Winter Conference High-Availability and Reliability, Bournemouth*, pages 25–26, February 2004.

[35] http://www.ultramonkey.org/papers/active_active/active_active.shtml. Saru: Active-active support for ultramonky.

[36] Ultra Monkey Project. http://www.ultramonkey.org/.

[37] Keepalived for Linux Linux High Availability. http://www.keepalived.org/.

[38] Piranha Load-Balanced Web and FTP Clusters. http://www.redhat.com/support/wpapers/piranha/.

[39] Tau Leng, Jenwei Hsieh, and Edward Yardumain. Architecting Linux High-Availability Clusters Part 1. *Dell Power Solutions*, 4:94–99, 2000.

[40] Elsayed A. Elsayed. *Reliability Engineering*. Addison Wesley, 1996.

[41] Larry L. Ball. *Cost-Efficient Network Management*. McGraw-Hill, 1992.

[42] Alan Wood. NonStop Availability in a Client/Server Environment. *Technical Report, Tandem Computers*, 1994.

[43] Paul Kales. *Reliability for Technology, Engineering and Management*. Prentice Hall, 1998.

[44] Divakara K. Udupa. *Network Management Systems Essentials*. McGraw-Hill, 1996.

[45] E.P. Duarte Jr., L.C.P. Albini, and A. Brawerman. An Algorithm for Distributed Diagnosis of Dynamic Fault and Repair Events. In *the Proceedings of the 7th IEEE International Conference on Parallel and Distributed Systems IEEE/CPADS'00, Iwate, Japan*, pages 299–306, 2000.

[46] http://www.horsburgh.com/h_npulse.html. "npulse project website".

[47] http://gicl.cs.drexel.edu/people/sevy/snmp/snmp.html. "snmp java package".

[48] http://www.mibble.org/. "mibble :: Mib parser".

[49] A. L. Robertson. Linux-HA Heartbeat Design. *Proceedings of the 4th International Linux Showcase and Conference, Atlanta*, pages 25–26, October 2000.

[50] Kishor S. Trivedi Robin A. Sahner and Antonio Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Using the SHARPE Software Package*. Kliwer Academic Publisher, 1996.

[51] Amit Singh. An Introduction to Virtualization. http://www.kernelthread.com/publications/virtualization/.

[52] Jeff Dike. User-mode linux. *Redhat Magazine*, November 2004. http://www.redhat.com/magazine/001nov04/.

[53] Linode Virtual Webservice Provider. http://www.linode.com/.

[54] Microsoft Virtual PC. http://www.microsoft.com/windows/virtualpc/default.mspx.

[55] Microsoft Virtual Server. http://www.microsoft.com/windowsserversystem/virtualserver/default.mspx

[56] VMware Workstation. http://www.vmware.com/products/desktop/ws_features.html.

[57] IBM z/VM Operating System. http://www.vm.ibm.com/.

[58] The User mode Linux Kernel Home Page. http://user-mode-linux.sourceforge.net.

[59] Linux Vserver. http://linux-vserver.org/.

[60] Microsoft Corporation. Microsoft virtual pc 2004 technical overview, http://www.microsoft.com/windows/virtualpc/evaluation/techoverview.mspx.

[61] Comparison of virtual machines. Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/comparison_of_virtual_machines.

[62] FAUmachine Project website. http://www3.informatik.uni-erlangen.de/research/faumachine/.

[63] Bills UML Root filesystems. http://www.stearns.org/mkrootfs/rootfs.html.

[64] UML Builder. http://umlbuilder.sourceforge.net/.

[65] UML File System Building Tool gBootRoot. http://freesoftwarepc.com/projects/gbootroot/.

[66] UML File System Building Tool rootstrap. http://http.us.debian.org/debian/pool/main/r/rootstrap/.

[67] User-Mode Linux Network. http://uml.openconsultancy.com.

[68] Networking UML Using Bridging. http://edeca.net/articles/bridging/index.html.

Vitae

- Louai Adnan Al-Awami.
- E-mail: louai@ccse.kfupm.edu.sa
- Born August 1979 in Qatif.
- Received Bachelor of Sciences in Computer Engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia in June 2002.
- MS in Computer Engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia in June 2006.
- Worked as a Graduate Assistant in the Computer Engineering Department at King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia between 2003 & 2006.
- Research interest includes Computer Networks Design, Network Management, Network Security, Wireless Networks, Multi-Valued Logic (MVL).
- Publications:
  - Abd-El-Barr, M. I. and Al-Awami, L. A., “Analysis of Direct Cover Algorithms for Minimization of MVL Functions”, The IEEE 15th International Conference on Microelectronics (ICM03), Dec. 9-11, 2003. pp. 308 312.

– Abd-El-Barr, M. I. and Al-Awami, L. A., “Iterative-based Minimization of Unary 4-valued Functions for Current-mode CMOS Realization”, Proceedings of the 34th International Symposium on Multiple-Valued Logic, 2004, 19-22 May, 2004. pp. 315 320.