Chapter 5

AN EM BASED SCALABLE AND RESPONSIVE NETWORK MANAGEMENT ARCHITECTURE FOR TELECOMM NETWORKS

5.1 INTRODUCTION

A typical organization model of a network management system is based on SNMP Client/Server architecture. It consists of two major components: network agent process and the network manager process. The network agent process resides on the managed network devices such as routers, switches, servers etc. The network manager is housed on the NMS station from where it manages the various devices, by accessing the management information, through the agents residing on them as shown in Figure 5.1. The management information consists of collection of managed objects, stored in Management Information Base (MIB).

![Figure 5.1: SNMP Client/Server architecture.](image)

Based on SNMP model, the management of networks from the Network Operation Centers (NOC) is mostly done by a Network Manager which deploys several element management system (EMS). EMSs in turn manage their specific zones or domains as shown in Figure 5.2. These EMSs provide ISO’s five functional categories: Fault Management, Configuration Management, Accounting, Performance Management and Security Management. Network Manager acts as a client to the EMSs, which in turn act as network manager to network devices for retrieval and provisioning of data. The data
retrieved by EMSs are stored in databases on the EMSs platforms and used for management of network devices.

**Figure 5.2: Network Management Architecture**

These existing management models traditionally adopt a centralized, Client/Server (C/S) approach wherein the management application, with the help of a manager, acting as clients, periodically accesses the data collected by a set of software modules, agents, placed on network devices by using an appropriate protocol.

These centralized architectures suffer from the lack of scalability and flexibility. Furthermore, the staleness of gathered data (due to network latency involved) and probable error in the selection of management task being carried over (owing to the staleness of data) reduces the reliability of the management applications.
Distributed management intelligence, a paradigm shift, offers a rational approach to overcome the limitations of centralised NM. Gathering and analysis of the management data from agents of managed devices is partitioned and spread over the various computing platforms in the network (sometimes managed devices act as computing platforms) thereby breaking the centralised paradigm. MA-based Network Management architectures offer an attractive solution in this paradigm for managing complex telecommunication and data networks.

However, models talk about deploying MA as mid-level managers on network devices for managing their own domains or zones. Many of these devices don’t support mobile agent runtime environment. Little attention has been paid on how to manage these devices without mobile agent execution environment like hubs and switches in enterprise networking environment and most of the telecomm devices like terminals, switches and multiplexers. This also includes legacy devices.

In view of the above mentioned limitations, this chapter presents a domain partitioned network management model based-on mobile agent & Element Management Systems in order to minimize management data flow to a centralized server. Intelligent agent allocated to specific EMS performs local network management and reports the results to the superior manager and finally the global manager performs global network management using those submitted management results.

As discussed in chapter 4, in client server model the management cost in terms of the data transferred to the Global manager for the whole network is directly proportional to the following factors:

1. Number of requests and responses to fetch the data remotely
2. Cost coefficient of the links on which information is exchanged.
3. The number of MIB accessed.
Whereas in the proposed model, EMSs manage the domains locally thereby minimizing the cost incurred due to costly inter-domain link traversal of mobile agents. The cost of managing flat bed model in IMASNM model is taken away as SQL interface will fetch the needed data from EMSs database and it will be kept in sync with network elements by publish/subscribe interfaces.

The proposed model discusses strategies for large scale network partitioning, fixing of management scope and deployment of mobile M-SNLMs in various sub-network domains. With a view to manage scalability the proposed model not only defines strategy to spawn child M-SNLM and their placement in sub-network domains but also the communication mechanism with higher level managers. Furthermore the management scalability has been improved by placing M-SNLMs close to the sub-network on the EMS platforms managing it and only returning needed management data to the supervisors.

In this sense, proposed management model based on mobile agents provides an attractive perspective to the lack of flexibility and scalability of current centralized management systems.

5.2 **HIGH LEVEL DESIGN OF EMS BASED NETWORK MANAGEMENT MODEL DEPLOYING MOBILE**

In this section we propose [88, 89] an architecture which provides a framework for Network Management functionality and related mobile agent management. Typical network management model based-on mobile agent works like this: management administrator dispatches corresponding mobile agents to managed nodes to perform different tasks, and these agents take back management information. Some shortcomings exist in this model like difficult to manage large-scale complex network, devices which do not support mobile agent run time environment escape from being managed and mostly it is incompatible with conventional SNMP based network management system.
Considering above, we proposed an EMS based mobile agent network management model shown in Figure 5.3.

Figure 5.3: EMS based MA distributed architecture.
In this scheme, the managed network is divided into many domains based on the geographical layout of the network or number of nodes a manager can efficiently manage or average load on the entire network or certain kind of administrative relationship. In each domain, an Element manager is appointed and for the overall managed network a set of various management applications along with a network manager acting as a Global Network Manager (GNM) are appointed. Element managers (EM) exist at the lowest level of the manager hierarchy. Each Element manager controls and monitors a set of network element (SONET/DWDM/Ethernet) via specialized protocol independent mediators as shown in Figure 5.4. They collect management data from these elements and store that in local databases.

![Figure 5.4. EMS architecture.](image-url)
After initial discovery of the network, EMS keeps track of database changes in the sub-network by means of publish/subscribe paradigm. Additionally a hierarchy of Mobile-Subnetwork Layer Manager (M-SNL) is appointed in between the GNM and leaf-level EMSs. First levels of M-SNLs (appointed by GNM) are dispatched to platforms where EMSs are running. M-SNLs along with EMSs take over a portion of the network from their parent M-SNLs and act as local managers for that portion of the network for all management needs. Depending upon the growth in their domain, M-SNLs spawn additional child M-SNLs for scalable management of the network. M-SNLs not only interact with the EMSs they are assigned to but also could visit other EMSs platforms depending upon the need as they are mobile agents in themselves.

5.3 ARCHITECTURE AND COMPONENTS

The proposed Network management Model based-on Mobile Agent can be seen as a four-tier management architecture:

1. Global Network Manager (GNM),
2. Mobile-SubNetwork Layer Manager (M-SNL),
3. Element Managers (EM) and
4. Managed Node (MN).

Mobile agents perform network management tasks in the system. M-SNLs main functions include mobile agent’s creation and dispatch as well as inter-subnetwork and element manager level events correlation M-SNLs run in agencies/servers which support mobile agents. JVM has been chosen as an execution environment for mobile agents. EMSs provide the platform for these agencies to run.

The main modules in GNM system are:

1. Management applications (configuration, fault, performance etc.),
2. Modules for handling defining management tasks in high level language,
3. Modules to delegate management tasks and
4. M-SNLM control mechanism (to initiate command and receive response).

M-SNLMs are arranged in a hierarchal manner. M-SNLMs download the management tasks from GNM dynamically and then delegate a part of or all management tasks to its child M-SNLMs. This is done as mobile agents. The run time environment to support M-SNLMs is present on all EMSs platforms. This is called mobile agent agency. Agency framework [93] includes two main modules:

1. Mobile Agent Creator/Dispatcher (MAC/D) and
2. Security module for mobile agents.

Agency determines the life cycle of mobile agents and MAC/D creates, dispatches, recalls and disposes mobile agents as needed. Agencies run on EMSs platforms. EMSs are typically three-tier architectures as shown in Figure 5.4.

EM services consist of management applications at EMS sub-network level. Communication & Session management establish the connection with network devices and management of communication session with them and mediation layer offers the advantage of protocol independence while communicating with the devices. Once EMSs discover and retrieve all the management data from the managed devices they establish Publish/Subscribe relationship with their domain elements and any change in their managed data is updated by EMSs in their database with the help of event channels working on publish/subscribe pattern.

5.4 DYNAMIC INTERACTION BETWEEN COMPONENTS

After getting dispatched from GNM or Supervisory/parent M-SNLM, typical interaction of a M-SNLM is shown in Figure 5.5. For a large & complex network, in between M-SNLMs (appointed by GNM) and Element Management Systems, there gets formed a hierarchy of supervisors (child M-SNLMs appointed by first level of M-SNLMs) which
interact with immediate next level subordinate M-SNLMs and EMSs assigned to them.

![Diagram](image)

**Figure 5.5: M-SNLM Visiting an EMS platform**

The management results are reported to Global NM via supervisors and Global NM is responsible for administrative sub-network partition, assigning tasks to each M-SNLM, indirectly, via supervisors and managing the whole network. Supervisors would maintain aggregated information such as available ports, wavelengths, states, fault reports and performance data about certain number of M-SNLMs and network management applications at NM layer would interact with these high level supervisory-managers instead of establishing direct connection with every network element. Provisioning operations (like Service, Equipment, LightPath) would be issues to high-level managers which then based on SubNetwork partitioning would delegate the information to subordinate managers. Monitoring of alarms and performance reports would operate in the reverse direction: M-SNLMs would aggregate it for their portion of the network and then would hand it over to their supervisors.
A management application would need to communicate with high-level supervisors in order to manipulate and monitor the entire network. In order to support the dynamic redistribution of the managers, so that they could adapt to the evolving nature of today’s networks, with rapidly changing traffic patterns and topology structures, all the managers are mobile agents in themselves. Communication between managers takes place via deglets [44]. A deglet is a lightweight agent with a transient task based life cycle model. There are two types of deglets: Provisioning deglets are the ones which flow from Supervisors (including NM) to Subordinates and upward flowing, EventReporting deglets, are those which report fault and performance events. Thus we have a hierarchically distributed deployment of cooperating mobile agents or “managers” which would lead to significantly reduced processing requirement at the Network Management level.

When a M-SNLM receives a request from GNM or its supervisor/parent M-SNLM, it formulates a set of subtasks for its subordinate/child M-SNLMs. Each subtask is transported to a M-SNLMs by a provisioning deglet. Upon reaching its target manager, each provisioning deglet attempts to perform the intended subtask. M-SNLMs interact with element managers to carry out the intended task. Each element manager immediately initiates a session with its corresponding physical device. The element manager then uses this session for transmitting commands and receiving notifications from the device. After interacting with the M-SNLMs, the deglet then encapsulates a report of the side effects and carries this back to the initiating manager. When all the deglets have returned, the manager aggregates the reports from below into a report for the original request. Collectively, provisioning deglets are referred to as control flow. A manager may send asynchronous notifications to its supervisor by using eventreport deglets. Eventreporting deglets encapsulate information about changes in the state of their sender. Upon reaching its target supervisor, each Eventreporting deglet attempts to notify the supervisor of the change in the subordinate’s state. The deglet then carries an acknowledgment of this notification back to the originating manager. Collectively, monitoring deglets are referred to as monitoring flow.
5.5 IMPLEMENTING AGENCY FOR MOBILE AGENTS ON EMS PLATFORM [93]

The agency provides the resources, execution environment, as well as communication support for Mobile Agents as discussed in chapter 3. The execution environment consists of Java interpreter called Java Virtual Machine, (JVM), which is a stand-alone platform. When the Mobile Agent arrives in the execution environment with the request to execute the agency first performs security operations upon the agent to ensure its safety and legality. Thereafter it is then instantiated and starts execution. After execution, the agency provides storage support for the intermediate results.

The SFC (Service Facility Component) component of mobile agent agency provides a supportive environment for service execution by the agent. It allocates the necessary resources required by an agent to perform its assigned task. For instance, if the mobile agent needs to run certain service like fault mgmt or configuration service, SFC provides the needed access to service instance and other resources. The problem with this arrangement is that it offers a static service access. The services are loaded by the same ClassLoader which loads other components of the agency. So if we have to load new services or upgrade existing services the agency has to shutdown and after including the change needs to restart. This causes the downtime and affects the live traffic condition. Section 5.6 discusses the augmentation required in Aglets SFC component to avoid this downtime and upload new services at runtime.

5.6 PROVIDING DYNAMICALLY CUSTOMIZABLE MANAGEMENT SERVICES

Most platforms for mobile agents are implemented in a monolithic way, with little or no provision for reconfiguration. This has several disadvantages –

1. The platform is hard to deploy, because a monolithic architecture does not integrate well with other applications.
2. It has poor or no extensibility. For instance, adding a new inter agent communication service to the system normally forces the whole platform to be recompiled and redeployed.

To address this limitation, a component based middleware for mobile agents based on Java is normally deployed. The components are structured as a set of JavaBeans [37], where a base component supports a core set of functionality such as: mobility, security and extensibility. Moreover, component can easily be added to any application, just like any other JavaBeans component, making it very simple to create mobile agent enabled applications. The functionality of the base component can be extended by plugging new services into its extensibility layer as shown in Figure 5.6. Services are also modeled as JavaBeans components. After being plugged, services became available to any client that can interact with the extensibility layer, including the application, the mobile agents and other services.

![Figure 5.6: Services offered as beans to extend mobile agent platforms](image-url)
Although this model has been used with success in implementing and deploying several services, like agent tracking, inter agent communication and disconnected computing, still it only supports static reconfiguration. The activities of adding, removing and upgrading services forces the application to be stopped and restarted, leading to a downtime which is undesirable in situations of live traffic carrying nodes. Thus there is a need to improve the model that could support dynamic reconfiguration. Services can be added, removed and upgraded at run time, with little or no disruption to the application or to the mobile agents that are running and using those services.

### 5.6.1 Dynamically Customisable Management Services

An architecture supporting dynamic reconfiguration must ensure that reconfigurations can be properly specified and carried out. In turn, this makes it desirable to build the middleware on top of a platform with support for dynamic linking and reflection. Java has good support for both features, which made it possible to implement dynamic services using only the standard interfaces available on the Java platform.

### 5.6.1.1 Requirements for Dynamic Reconfiguration

As identified in [83], there are three basic issues that must be addressed by any system that supports dynamic reconfiguration:

- **Specification and management of change** There must be a way to describe the changes that are to be made and the conditions required by them. This includes specifying software and hardware requirements for a component, and the external relations and bindings with other components.

- **Preservation of Consistency** After reconfiguration, the system should be working properly. The various components should remain in a mutually consistent state and no bindings should be broken.

- **Minimum disruption** The objective of dynamic reconfiguration is to avoid disruption of the service provided by the application especially those which are not being reconfigured should not be disrupted.
This work proposed here focuses only on the last two points, i.e., supporting upgrades with minimal disruption to the system, while preserving its consistency.

5.6.1.2 Role of Java platform towards creating Dynamic Reconfiguration environment

Java platform is the most suitable platform for the creating of dynamic reconfigurable environment owing to its following features:

1. Lazy loading. Classes are loaded on demand. Class loading is delayed as long as possible, reducing memory usage and improving system response time.

2. Type-safe linkage. Dynamic class loading must not violate the type safety of the Java virtual machine. Dynamic loading must not require additional run-time checks in order to guarantee type safety. Additional link-time checks are acceptable, because these checks are performed only once.

3. User-definable class loading policy. Class loaders are first-class objects. Programmers have complete control of dynamic class loading. A user-defined class loader can, for example, specify the remote location from which the classes are loaded, or assign appropriate security attributes to classes loaded from a particular source.

4. Multiple namespaces. Class loaders provide separate namespaces for different software components. For example, the HotJava™ browser loads applets from different sources into separate class loaders. These applets may contain classes of the same name, but the classes are treated as distinct types by the Java virtual machine.

More information on this aspect of Java related to dynamic class loading is given in Appendix A.
5.6.2 Design of Dynamic Service Manager

As discussed in chapter 3, the SFC (Service Facility Component) component of mobile agent agency provides a supportive environment for agent execution. It allocates the necessary resources required by an agent to perform its assigned task. For instance, if the mobile agent needs to run certain service like fault mgmt or configuration service, SFC provides the needed access to service instance and other resources. The problem with this arrangement is that it offers a static service access. The services are loaded by the same ClassLoader which loads other components of the agency.

As a result the services can be stopped but it is not possible to unload its classes without unloading the whole agency. It may be noted that the same class loader should not dynamically load and unload services at run time. For example the services 1, 2 and 3 shown in Figure 5.7 already loaded in the same class loader through a static Service Manager cannot be unloaded without shutting down the application first.

![Diagram showing static service provision](image-url)
Therefore to provide dynamic loading and unloading of services the Static Service Manager is replaced by a Dynamic Service Manager. The Dynamic Service Manager maintains a reference to a ClassLoader (similar to one shown in Figure 5.6). Whenever a mobile agent arrives on an agency, it dynamically loads a service that needs to be run. In fact a separate class loader is used for loading each service as shown in Figure 5.8.

Thus the various services have been isolated from each other, making them independent units. It is even possible to execute different versions of a service side by side much needed when different versions of the service are not backwards compatible. Additionally the services have been augmented with following features towards achieving dynamic provisioning.

A ServiceDescriptor An object used to describe a service, defining its identity. It contains the name, version and the service interface name. Two services are equal if their
service descriptors are equal. Two services are compatible if they have the same name and service interface.

**A service interface and its implementation.** It is how the clients interact with a service, and defines the functionality specific to the service. This means that each service will have a specific interface that is known by the clients. The service must also provide an implementation of this interface. This allows services to be upgraded, as it is explained below.

### 5.6.2.1 Dynamic loading and Upgrading of Services

For loading a new service, the DynamicServiceManager needs to know where the Jar file with the service is, and what is the name of the service class that implements a given service interface. It will create a new class loader for this service, using it to load and instantiate the service provider. Figure 5.9 illustrates that a DynamicServiceManager class can redirect the service request to new version of Service class.

![Figure 5.9: DynamicServiceManager redirects to new version of service class](image)

The key point here to note is that the classloader must load all three classes i.e. DynamicServiceManager, old service class and new service class into separate class loaders.
The code snippet for the DynamicServiceManager for upgrading services is shown in Figure 5.10.

```java
class DynamicServiceManager {
    private Object Service;

    public void upgradeService (String location) {
        MyClassLoader cl = new MyClassLoader(location);
        Class c = cl.loadClass(“Service”);
        Service = c.newInstance();
    }

    public void processRequest(…) {
        Class c = service.getClass();
        Method m = c.getMethod(“run”, …);
        m.invoke (service, …);
    }
}
```

**Figure 5.10: DynamicServiceManager for upgrade service**

The DynamicServiceManager.processRequest() method redirects all incoming requests to a Service object stored in a private field. It uses the Java Core Reflection API [37] to invoke the “run” method on the service object. In addition, the Server.upgradeService() method allows a new version of the Service class to be dynamically loaded, replacing the existing service object. Callers of upgradeService() supply the location of the new class files. Further requests are redirected to the new object referenced to by service.

**Upgrading of Services through a service interface**

As discussed earlier that it is through service interface a client interacts with a service, and defines the functionality specific to the service. This means that each service will have a specific interface that is known by the clients. The service must also provide an implementation of this interface as shown in Figure 5.11.
“NMSServiceInterface” is the interface through which all the services are accessed in DynamicServiceManager model.

![Diagram](image)

**Figure 5.11: Interface for upgrade service**

Since the implementation has been separated from the interface it can be easily upgraded. The code snippet for the DynamicServiceManager for upgrading services with a service interface is shown in Figure 5.12.

```java
class DynamicServiceManager {
    private NMSServiceInterface Service;
    public void upgradeService(String location) {
        MyClassLoader cl = new MyClassLoader(location);
        Class c = cl.loadClass("Service");
        Service = (NMSServiceInterface) c.newInstance();
    }

    public void processRequest(...) {
        Service.run();
    }
}
```

**Figure 5.12: DynamicServiceManager for upgrade service**

Reflecation [37] allows DynamicServiceManager class to use the Service class without a direct reference. Reflection is a feature of the Java platform that makes the internal
structure of a class or interface available to the programmer. This meta-information includes, among other things, the names of the methods, of the fields, interfaces implemented and super-classes. This is important for supporting dynamic reconfiguration, allowing a framework to interact at run time with components that are unknown at compile time.

Alternatively, DynamicServiceManager and Service class can share a common interface, here, NMSServiceInterface, as shown above. Dispatching through an interface is typically more efficient than reflection. The interface type itself must not be reloaded, because the DynamicServiceManager class can refer to only one NMSServiceInterface type. The getServiceClass() method must return a class that implements the same NMSServiceInterface every time.

Once the updateService() method is called, all future requests are processed by the new Service class where it may be possible that the old Service class may not have finished processing some of the earlier requests giving room for two Service classes may coexist for a while. This continues till all requests on old class are complete and thereafter all the references to the old class are dropped and it is unloaded.

5.6.3 Unloading a service
To unload a service it is necessary to
1. release all references to instances of the service’s classes,
2. objects of the type Class representing classes from the service, and
3. finally the service class loader.
When this happens, the garbage collector is free to release the class loader and any classes defined by it.

5.7 HIGHLIGHTS OF THE PROPOSED MODEL

EMS based domain partitioned network management system based-on mobile agent has following technological advantages over network model such as conventional network
management system and other MA based network management models

1. Earlier work presented by researchers [70][86] assumed the presence of mobile agent runtime environment on the network devices. This may not be true for many telecomm elements and legacy devices. Proposed model make the architecture independent of this constraint.

2. Earlier models [70][86] dispatch data agents to devices in the sub-network and bear a cost associated with it. Proposed model saves this cost by retrieving that data from the database present on the EMS platform only. It also controls the number of mobile agents present in the system as a given M-SNLM can not only manage its assigned domain but can also visit other domains and interact with EMSs to manage the network.

3. Bottleneck problem of convention centralized network management are addressed and static communication problems in distributed network is solved efficiently. The basic concept which is adopted in this framework is: “M-SNLMs solve problems locally and the supervisors/parent M-SNLMs worry about the issues which are propagated to them.”

The proposed architecture based on mobile agent technology has the following advantages:

1. Balancing of Network loads,
2. The framework is adaptive with network size change,
3. Fault-tolerant nature of MA-based design,,
4. Proposed model offers a hierarchical network management model that not only reduces network management complexity but also minimize the management data transferred to and fro in the network.

5.8 PERFORMANCE ANALYSIS

Many a times there arises a situation in network monitoring activities where one or two MIB variables representing some managed object may not be able to give good
assessment of the issue (like performance degradation, signal failure, packet loss etc.). However, we need to access multiple parameters and then make an assessment. In order to carry out the performance analysis of the model presented in this work a bandwidth and throughput utilization function known as health function (HF) [80] which is essentially an aggregate of multiple MIB variable. In this five MIB-II [23] managed objects are combined to define the percentage $E(t)$ of IP packets discarded over the total number of packets sent within a specific time interval as shown in Equation (5.1)

$$E(t) = \frac{(ipOutDiscards + ipOutNoRoutes + ipFragFails)}{(ipOutRequests + ipForwDatagram)} \times 100 \quad (5.1)$$

In simple client/server based SNMP model, either we have to issue five get requests to retrieve the values or at the best we would issue a single get request and the response would carry all the five parameters with their value. Back at manager level the value of $E(t)$ would be computed and necessary action would be planned. As suggested by Damianos Gavalas et al.[70] MAs can compute HFs locally thereby providing a way to semantically compress large amount of data (five variables, their IODs and return values and other SNMP Req/Res overheads) in a single value returned to the manager, thereby relieving it from processing NM data, while MAs state size remains as small as possible. But in case we have to compute this value for a collection of nodes, MAs would have to travel to these nodes and we need to bear the cost presented in [86].

The proposed mechanism offers an EMS database solution where we could issue a simple SQL query so that the needed values could be computed in a far less time and far less cost. For example, the $E(t)$ function shown in Equation (5.1), a simple SQL query similar to shown below can easily compute the function value at database level itself. With appropriate measures taken as schema design level like (indexes and data arrangement) similar SQL across series of nodes could be carried out at fraction of the cost that a mobile agent based flat bed model would incur.
Consider the HealthFunction given below as a view or some database schema table

<table>
<thead>
<tr>
<th>ipOutDiscard</th>
<th>ipOutNoRoutes</th>
<th>ipFragFails</th>
<th>ipOutRequest</th>
<th>ipForwDatagram</th>
</tr>
</thead>
</table>

HealthFunction Table/View

The corresponding SQL query would be as shown below:

```sql
SELECT ipOutDiscards + ipOutNoRoutes + ipFragFails AS “Packet Discarded”,
ipOutRequests + ipForwDatagram AS “Total Packets”,
Packet Discarded / Total Packets AS “Health Function”
From HealthFunction
Where NE = `xxxx`;
```

### 5.9 NETWORK MANAGEMENT COST CALCULATION FOR THE PROPOSED MODEL

The Network management cost for IMASNM Model is discussed in details by A.K. Sharma et. al. [86] and has been used here as basis for cost calculation for the proposed model. The network management cost calculation for the proposed model as shown in Equation (5.2) involves not only the management traffic cost i.e. (messages between the managers and sub-domain managers) but also the cost of setting up the managers as per the initial discovery of the network.

\[
C_{\text{TOTAL}} = C_{\text{SETUP}} + C_{\text{MGMTTR}} \quad \ldots \ldots (5.2)
\]

Where

- \(C_{\text{SETUP}}\): cost for discovering the network and deploying the managers – both M-SNLM & EMS.
- \(C_{\text{MGMTTR}}\): cost of a typical polling to know whole network status at top most level.
C_{SETUP}, the cost for discovering the network and deploying the managers – both M-SNLM & EMS is computed as shown in Equation (5.3)

\[
C_{SETUP} = C_{MSNLM} + C_{EMS} \quad \ldots \ldots (5.3)
\]

The cost of setting up the top level M-SNLMs managers as per the initial discovery of the network would be computed as shown in Equation (5.4)

\[
C_{M-SNLM} = \sum_{h,j}^{h,j} M_{\text{Size}} \quad \ldots \ldots (5.4)
\]

Where

- \(M_{\text{Size}}\): Size of mobile manager,
- \(L\): Number of mother manager in the network,
- \(M\): Number of child managers in the sub domains of \(h^{\text{th}}\) mother manager and
- \(F_{h,j}\): Sum of all the link cost coefficient between \(h^{\text{th}}\) mother manager to \(j^{\text{th}}\) child manager’s sub domain.

The element managers would discover the domain element details in client/server mode and store the data in the EMS platform data for the agents visiting their platforms. The cost for setting up such platforms across the network would be computed as shown in Equation (5.5)

\[
C_{EMS} = \{ 0,i *(S_{\text{req}} + S_{\text{res}}) \} * M * L \quad \ldots \ldots (5.5)
\]

Where

- \(N\): the no. of devices under the management of a given EMS,
- \(L\): Number of mother manager in the network,
- \(M\): Number of child managers in the sub domains of a mother manager and

It may be noted that the deployment cost of the proposed model is higher than the IMASNM Model as cost of EMS gets added to the mobile-manager deployment. But as
suggested in the proposed model architecture that it is a onetime cost and once the EMSs discover the network. They keep the data fresh through publish/subscribe interface. So for long term polling operations this cost could be ignored.

As discussed in chapter 4, for IMASNM [86] model the management traffic cost for various polling operations is given as shown in Equation (5.6)

\[
C_{MGMTTR} = h_{ij}(MA_{res}) + Q \hspace{1cm} \text{(5.6)}
\]

Where

\(MA_{res}\): Size of message sent by child to mother manager to report network health of its underlying domain,

\(C_Q\): Flat bed model cost for \(Q^{th}\) domain’s M-SNLM.

\[
C_Q = MDA_{\text{Size}}*(R_Q+1)*K_Q \hspace{1cm} \text{(5.7)}
\]

Where

\(MDA_{\text{Size}}\): Size of mobile data agent,

\(R_Q\): number of managed node is \(Q^{th}\) domain and

\(K_Q\): link coefficient of link’s of \(Q^{th}\) domain.

It may be noted that the flat bed cost increases in proportion to the cost coefficient of the links as well as number of nodes being managed in a given sub-domain. In the proposed model, the SQL performed by M-SNLMs would be local only and the cost incurred by flat bed model could be saved in total.

Hence, under the assumption that the cost of sql query would be far less than the cost of managing a domain by flat-bed model approach, the polling cost of the entire network could be computer as shown in Equation (5.8)

\[
C_{MGMTTR} = h_{ij}(MA_{res}) \hspace{1cm} \text{(5.8)}
\]
Ignoring the one time setup cost, Equation (5.2) could be shown as

\[ C_{\text{TOTAL}} \approx C_{\text{MGMTTR}} \]  
(which is already optimised)

---

**Figure 5.13: Experimental Setup for MA based network management model using MYSQL Database**

A network management cost comparison between the MA based flat-bed model and EM base model which offers a database for SQL query purpose is carried out with a modified setup of chapter 3 as shown in Figure 5.13.
Table 5.1 and Figure 5.14 show the timing values for three nodes i.e. Node1, Node2 & Node3 through a simple SQL query.

Table 5.1: Experiment result of Mobile agent based network model accessing MYSQL database

<table>
<thead>
<tr>
<th>S. no</th>
<th>Time1</th>
<th>Time2</th>
<th>Time3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>122</td>
<td>39</td>
<td>31</td>
</tr>
<tr>
<td>2.</td>
<td>47</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>3.</td>
<td>143</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>4.</td>
<td>31</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>5.</td>
<td>16</td>
<td>31</td>
<td>15</td>
</tr>
<tr>
<td>6.</td>
<td>30</td>
<td>23</td>
<td>18</td>
</tr>
<tr>
<td>7.</td>
<td>27</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>8.</td>
<td>44</td>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>9.</td>
<td>32</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>10.</td>
<td>16</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>44</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>12.</td>
<td>35</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>13.</td>
<td>30</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>14.</td>
<td>30</td>
<td>27</td>
<td>20</td>
</tr>
<tr>
<td>Average</td>
<td>46</td>
<td>30</td>
<td>25</td>
</tr>
</tbody>
</table>

Figure 5.14: Graph of table 5.1 data values

Above experimental results shows that mobile agent based NM takes far less time as compare to client server based NM as well as MA-based flat-bed model. This simple experimental result shows that proposed EM based model would scale much better than Client/Server as well as IMASNM model.
5.10 SPANNING TREE BASED TRAVERSAL PLANNING FOR MA BASED NETWORK MODELS

As discussed above, the MA based NMS is better than CS based NMS. But the itinerary planning for MA based NMS can further improve the bandwidth utilization.

When MA follows an arbitrary route, it is very likely that it traverses on the link having a high bandwidth utilization and greater round trip time thereby increasing the overall cost of bandwidth utilization. Whereas a carefully planned itinerary for MA based network management can improve the bandwidth utilization and round trip time.

In this work a spanning tree based mechanism is being proposed that improves the itinerary for mobile agents in MA based network management model. The working of the proposed technique is given below.

- Here, a network is represented as an undirected and connected graph, say G, consisting of nodes (V) and edges (E) representing managed devices and communication links respectively. It can be represented as \( G = (V,E) \)
- In order to discover an MST (minimum spanning tree) for G, Prim’s Algorithm is applied. The algorithm for discovery of MST is given below

**Algorithm MST(G) {**

1. Choose any element \( r \); set \( S = \{r\} \) and \( A = 0 \) (Take \( r \) as the root of spanning tree, \( S \) as the set of vertices in MST and \( A \) as the set of edges in MST)
2. Find the lightest edge, in terms of weight given to an edge (here, bandwidth of the link), such that one point is in \( S \) and the other is in \( V \setminus S \) (all the vertices not in \( S \)). Add this edge to \( A \) and its (other) endpoint to \( S \).
3. If \( V \setminus S = 0 \), then stop and output the minimum spanning tree (\( S,A \)). Otherwise, goto step 1.

**}
Consider an undirected weighted graph having 7 nodes (V1 to V7) and MA starts traversing from node V1 and traverse all the nodes (V2 to V7) to collect data from each node as shown in Figure 5.15.

Along the edges the weights (cost coefficient) shown essentially indicate the traversal cost which is indirectly proportional to bandwidth of the edge/link. The cost for network management not only depends on management data size but also on cost coefficient of network link through which management data pass. To calculate the cost of the MA traversal all through the network assigned to it, following cost calculation considerations were carried out.

5.10.1 Itinerary for Mobile Agents

1) When MA follows Arbitrary Itinerary:

The MA starts traversing graph from node V1 and follows the path from V1-V7-V6-V5-V4-V3-V2 as shown in Figure 5.16. The total cost coefficient of path is calculated as \((6+4+2+3+6+5=26)\).
2) **When MA follows MST Based itinerary:**
When the MA follow MST based itinerary as shown in Figure 5.17. The MA starts traversing graph from node V1 and follows the path from V1-V6-V5-V4-V7-V2-V3 as shown in Figure 5.17. The total cost coefficient of path is calculated is \((1+2+3+3+1+5=15)\).
Thus in terms of cost coefficient of path, MST based itinerary is better than arbitrary itinerary.

The adjacency matrix of undirected weighted graph is given in Figure 5.18.

![Adjacency matrix of undirected weighted graph](image)

**Figure 5.18: Adjacency matrix of undirected weighted graph**

The proposed mechanism takes adjacency matrix of weighted graph as input and returns the minimum spanning tree as an output.

From cost computations done in earlier in chapter 3, Cma=3272 Bytes

**Case A:** Non MST (Arbitrary) Route, Total Cost Coefficient = (6+4+2+3+6+5=26)

Management Cost Cma= 3272*26 =85072 Bytes
Case B: MST Based Route, Total Cost Coefficient = (1+2+3+3+1+5=15)
Management Cost Cma= 3272*15 = 49080 Bytes
The management cost computed for number of times the polling is done is tabulated in Table 5.2.

Table 5.2: Management Cost of Non MST vs. MST Based NMS

<table>
<thead>
<tr>
<th>Number of Polling</th>
<th>A(Non-MST)</th>
<th>B(MST)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>85072</td>
<td>49080</td>
</tr>
<tr>
<td>5</td>
<td>425360</td>
<td>245400</td>
</tr>
<tr>
<td>10</td>
<td>850720</td>
<td>490800</td>
</tr>
<tr>
<td>20</td>
<td>1701440</td>
<td>981600</td>
</tr>
<tr>
<td>50</td>
<td>4253600</td>
<td>2454000</td>
</tr>
</tbody>
</table>

Figure 5.19: Path Followed by Mobile Agent for MST Itinerary

It may be noted from Table 5.2 that with the increase in number of polls the management cost of non-MST based traversal increases many fold as compared to MST based traversal model as also illustrated in Figure 5.19.
5.11 Conclusion

Mobile agents offer an easy re-configurable, flexible and scalable solution to the management of today’s complex telecommunication networks thereby reduces the number of necessary human interactions. Many of the complex management tasks can be delegated to agents whom agents can easily carry out without much intervention from the higher management layers. As discussed in the mathematical model analysis, the independence and mobility of mobile agents reduce bandwidth overloading problems by moving a processing of the management data and decision making from centralized management stations to the managed devices thereby saving many repetitive request/response roundtrips and also address the problems created by intermittent or unreliable network connections between the network management stations and managed devices. Agents can easily work off-line and communicate their results when the application is back on-line. Moreover agents support parallel execution (load balancing) of large computation which can be easily divided among various computational resources. Thus using agents network monitoring and other management tasks can be easily decentralized.