5. Research Issues Addressed.

There are good arguments in support of the claim that agent technology will prove to be a valuable tool for building complex distributed systems. But as yet, these arguments are unsupported by much substantial evidence: agent-technology is essentially immature and untested. With no experience to guide them, agent system developers tend to find them falling into the same traps. The various issues addressed are discussed below.

5.1. Resource Discovery:

Current network management (NM) systems are typically designed according to a centralized NM paradigm characterized by a low degree of flexibility and re-configurability. Management interactions are based on a centralized, client/server model, where a central station (manager) collects, aggregates and processes data retrieved from physically distributed servers (Here agent refers to the static management agent and is different from mobile-agents.). Widely deployed NM standards, such as the Simple Network Management Protocol (SNMP) are designed according to this rigid centralized model. Within this protocol, physical resources are represented by managed objects. Collections of managed objects are grouped into tree-structured Management Information Bases (MIB). The centralized approach in NM is known to present severe efficiency and scalability limitations because of the reasons stated below:

- The process of data collection and analysis typically involves massive transfers of management data causing considerable strain on network throughput and
- Processing bottlenecks at the manager host.

All these problems suggest distribution of management intelligence as a rational approach to overcome the limitations of centralized NM. The Internet Engineering Task Force (IETF) has proposed an approach, known as RMON (Remote MONitoring), which introduces a degree of decentralization. RMON monitoring devices (probes) collect management statistics from their local domain (e.g. an Ethernet segment), providing detailed information concerning traffic activity.
In terms of recent research activities, Management by Delegation (MbD) represents a clear effort towards decentralization of management logic. The initial approach was to download management scripts that were compiled and executed at the agent side. However, the advent of Java has made this task significantly simpler.

Managing host resources (disk space, services running, memory usage, etc.) is an important part of network management. The distinction between traditional system administration and network-management has been disappearing over the last decade, as Sun Microsystems puts it as "The network is the computer". The Host Resources MIB (RFC 2790) defines a set of objects to help in managing critical aspects of UNIX and Windows systems. Some of the objects supported by the Host Resources MIB include disk capacity, number of system users, number of running processes, and software currently installed. In today's e-commerce world, more and more people are relying on service-oriented web sites. Making sure your backend servers are functioning properly is as important as monitoring your routers and other communications devices.

The Host Resources MIB provides many types of functions, primarily in the fault-management, configuration and asset-management and performance-management areas. The Host Resources MIB helps a number of system management jobs that a network manager might face. With this MIB module, a network manager can download an inventory of all equipment on various LANs across an organization, without regard to what types of systems the equipment is attached to. In addition to determining how much memory and disk is installed in each computer, the types and versions of other hardware and software components can be retrieved. Obsolete versions of software or hardware can be flagged and incompatibilities between various hardware and software components can be detected. Disk drives can be monitored to make sure that routine backup procedures are being followed and that the disks are not running out of space.
Proposed system:

Here the system that uses a mobile-agent technology (a form of decentralized approach as compared to existing centralized client/server based SNMP systems) for discovering the resources available on hosts connected over a network is proposed. Agents can realistically be of benefit in areas concerned with autonomy and mobility. This is especially true of network monitoring applications and this will be the focus of this work. The usage of mobile-agents and the advantages that these have over traditional client/server applications are also discussed subsequently. Mobile-agents' capability of moving to remote nodes and their ability to perform the assigned tasks in these remote nodes has resulted in more attention being paid in performing real-time network monitoring.

Investigation of suitability of application of the mobile-agent technology for such systems involves issues like:

- Network Latency and Round Trip Time.
- Transfer of huge data.
- Tasks that involve complex computations to be performed on huge data sets.
- Tasks that are inherently parallel and distributed in nature.

and are discussed later in this section.

Automatic mechanisms to extract resources available in computer network using the mobile-agents are found to be much better in terms of the performance. Figures 5.1, 5.2 and 5.3 illustrate the system/framework developed. Knowledge about resource, their usage, usage patterns, may be developed for the administrator to know what is happening and how things are being used, for planning the future. There are many issues about the mobile-agents while building such applications and are discussed below.
Figure- 5.1: Use of Mobile-Agents in NM.
Figure 5.2: GUI of System Developed for Resource Discovery.
<table>
<thead>
<tr>
<th>No.</th>
<th>Name of the Parameter</th>
<th>Old Value</th>
<th>New Value</th>
<th>Recent Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System Name</td>
<td>NODE220</td>
<td>NODE220</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Date</td>
<td>UNX Epoch Date is 01/01/1970 05:39 AM Today is 1708/02/2004 04:19 PM</td>
<td>UNX Epoch Date is 01/01/1970 05:39 AM Today is 1708/02/2004 05:16 PM</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>System Uptime</td>
<td>0 hours, 31 minutes, 51 seconds.</td>
<td>1 hour, 38 minutes, 32 seconds.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Number of Users</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Processor Load</td>
<td>89% of the time processor busy</td>
<td>59% of the time processor busy</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Physical Memory</td>
<td>130552 KB</td>
<td>130552 KB</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Storage Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Device Description</td>
<td>Intel MS TCP Loopback interface NDIS 5.0</td>
<td>Intel MS TCP Loopback interface NDIS 5.0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Services Running</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Software Installed</td>
<td>Cambasia Studio 2 * Microsoft Internet Explorer 6</td>
<td>Cambasia Studio 2 * Microsoft Internet Explorer 6</td>
<td></td>
</tr>
</tbody>
</table>

Figure- 5.3: Results of System Developed for Resource Discovery.
5.2. Effectiveness of mobile-agents - an issue:
The effectiveness of mobile-agents for distributed computing applications was done as a set of experiments using IBM’s AGLET and in JADE platform and observations are as under:

1. Agents are not suitable for data-transfer as it is done in conventional systems.
2. Agents are better suited for applications where the job is dispatched to carry upon the remote data rather than bringing the data and performing the computations.
3. Mobile-agents are not enough matured to perform distributed computing applications.

Experiments were conducted in ethernet based LAN with 10/100 Mbps Intranet and 1Mbps leased line internet connectivity. Twenty PIV machines of 1.4 GHz with 128 MB RAM, Linux version 9.0, Windows XP Operating System, JDK 1.4.1, IBM Aglet Software Development Kit 2.0.2 and Jade Platforms were used.

5.2.1 Experiment- 1 (Variation of RTT w.r.t size of agent): Here, the agent was made to travel through several hosts and round trip time (RTT) was measured. The following observations were made:

- As the size of the agent increases, the network latency and RTT also increase minimally.
- Agent fails to dispatch at 6291455 bytes. The reason being the constraints of JVM on the size of the object.

This imposes the restriction on the size of the knowledge that the agent carries with it during migration from one host to another. The result obtained for the two agent platform under no load and full load conditions for ‘Jade’ and ‘Aglet’ are shown in figures 5.4 and 5.5.
Comparision of JADE vs. Aglet (Full load Condition)

Agent size in MB

Figure-5.5: Comparison of Performances of JADE and Aglet under Full Load Condition.

 Comparision of JADE vs. Aglet (No Load Condition)

Agent size in MB

---→Java.lang.OutOfMemoryError /Aglet failed to dispatch

Figure-5.5: Comparison of Performances of JADE and Aglet under Full Load Condition
5.2.2. Experiment-2 (Performance of Jade, Aglet and FTP for file transfer): Here the file of different size was transferred over a network using traditional File Transfer Protocol (FTP) and also using agent and the time was measured as shown in figures 5.6 and 5.7.

The following observations were made:

- Agents are not suitable for huge data transfers.
- The poor performance of the mobile-agents is due to the object serialization process, in which object is marshaled and unmarshalled.
- This justifies the statement that, when very large volumes of data are stored at remote hosts, these data should be processed in the locality of data rather than transferring over the network.
- If the chunk size of the agent is increased, agents perform better than FTP.

---

![Diagram: Comparison of FTP, JADE, and Aglet](image)

**Figure-5.6:** Performance of Jade, Aglet, and FTP for File Transfer.
5.2.3. Experiment- 3 (Performance of Aglet involving huge computations): Here the suitability of application of mobile-agents in applications that involves huge computations was evaluated as shown in figure 5.8.

Case I: Refers to the case where the agent goes to the remote data, performs computations and brings the result back.

Case II: Refers to the case where the agent brings the remote data, performs computations locally and obtains the result.

The following observation was made:

- When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather than transferred over the network; The motto is simple: move the computations to the data, rather than the data to the computations.
Use of Aglets for applications involving huge computations

![Graph showing Performance of Aglet Involving Huge Computations.](image)

Figure-5.8: Performance of Aglet Involving Huge Computations.

Case-I: Agent (computational logic) moves to the job (Huge Data).

Case-II: Agent gets the data and performs the job. i.e. data is moved to computational logic.

Experiment conducted on LAN, File Size: 1 Lac elements.

5.2.4 Experiment- 4 (Performance of Jade and Aglet for distributed searching): Here the study was made to investigate the effectiveness of using a mobile-agent in searching for a key element in a network and is compared with the traditional searching approach. The results are shown in figure 5.9.

The following observation was made:

- Mobile-agent technology is a viable and suitable approach for applications that involve distributed searching.
- Parallel execution of agents for searching shows a uniform performance in execution time.
5.2.5. Experiment-5 (Distributed sorting): Here, an investigation about the effectiveness of using a mobile-agent in sorting a huge data file (centrally available and distributed to various machines using agents to use their computing resource) was done and its performance behavior was compared with the traditional sorting approach as shown in figure 5.10.

The following observation was made:

- As the data subsets are dispatched to remote nodes for sorting, the marshalling and unmarshalling takes time.
- Hence we observe that distributed sorting using an agent is not a viable approach.

Figure-5.9: Performance of Jade and Aglet for Distributed Searching.
Figure-5.10: Comparison of Performance of Mobile-Agents (Aglet & Jade) with Traditional Sorting Techniques.

5.2.6. Experiment-6 (Distributed data mining): Here, the effectiveness of mobile-agents for mining association rules in real-time distributed environment were addressed and performances were compared with the traditional client server approach. Over the past decade, data mining has gained an important role in analysis of large datasets and there by understanding the complex systems in almost all areas. Such datasets are often collected in a geographically distributed way, and cannot, in practice, be gathered into a single repository. The existing data-mining methods for distributed data are communication intensive. Many algorithms for data-mining have been proposed for data at a single location and some at multiple locations with improvement in terms of efficiency of algorithms as a part of quality but effectiveness of these algorithms in real
time distributed environment are not addressed, as the data on the web/network are distributed by its nature. As a consequence, both new architectures and new algorithms are needed. In section the software agent technology was introduced to support the building of distributed data-mining architecture and capabilities of mobile-agents' paradigm were explored to show by an experiment that, it is suited for distributed data-mining compared to traditional approaches like client server computing.

Cumbersome computations can often be broken into discrete units for distribution among a pool of servers or processors. Each of these discrete units can be assigned to an agent, which is then dispatched to an "agent farm", where the work is actually performed. Upon completion, each agent can return home and the results can be aggregated and summarized.

The goal of this work is to develop a system using mobile-agents and show quantitatively how mobile-agents are better to mine the huge data available in the distributed environment where the component data are distributed among several sites, namely D1, D2, D3, DN; having same format, Tj={i1,i2,...,im}, where Tj is transaction identifier and ik(1<=k<=m) is unique items in the transaction. If D is considered as central data which is specified implicitly with the help of distributed data among several sites then D=D1 U D2 U...DN, then the association rules are to be generated on implicit dataset D by generating them remotely on distributed sites and importing results to the central site.

The entire problem can be defined as follows:

\[ R = F_{s,c}(D) \]

Where:

- 'F' is the Data mining function (Ex: Association Rule) with minimum support 's' and confidence 'c'.
- 'R' is final set of association rules generated.
- 's': is minimum support value for generation of association rules over the available dataset.
- 'c': be the minimum confidence value.
Support(s) and Confidence(c) can be defined as below:

Let 'N' be the number of transactions in the transaction database. Let 'Txy' be the set of large transactions that contain a set of extended-items 'x U y' and 'Tx' be the set of large transactions that contain 'x'. Then, support and confidence of a transaction association rule  $x \Rightarrow y$ are defined as

$$\text{Minimum Support(s)} = \frac{T_{xy}}{N}$$

$$\text{Confidence(c)} = \frac{T_{xy}}{T_x}$$

This can be decomposed to apply to the distributed component data as:

$$R = G[fs,c(D_1), fs,c(D_2) \ldots fs,c(D_N)]$$

Where:

- 'G' is function, which consolidates the final rules obtained from distributed site.
- 'fs,c' is an Apriori or any algorithm applied at each distributed site.

Finally, 'R' contains the union of all the association rules on explicit datasets available at remote sites.

$$R = \bigcup_{i=1}^{N} (Fs,c(D_i))$$

The GUI of the system developed is shown in figure-5.11.
The experiments were conducted using scatter-gather style. In the first case agents were sent to remote data repository and large datasets were transferred to the central site and then data-mining algorithm was applied on this collected large dataset at central site. The performances observed are shown in figure-5.14 and table -1. Around 6MB data size was used for the faster data transfer because as data chunk size is increased the time taken by mobile-agent to migrate reduces, as indicated in figure-5.11.1 and table-0. In the second case agents were sent to each remote data repository. Each such agent computes local models, and were combined in to a global model at the central site. The performances observed are shown in figure-5.15 and table -1. The comparative performances of these two cases are shown in figure-5.16.
### Table-0: File Transfer Using Mobile Agents with Varying Chunk Size.

<table>
<thead>
<tr>
<th>Chunk size</th>
<th>Transfer time (in msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>169804</td>
</tr>
<tr>
<td>1024</td>
<td>75619</td>
</tr>
<tr>
<td>2048</td>
<td>40899</td>
</tr>
<tr>
<td>4096</td>
<td>31405</td>
</tr>
<tr>
<td>8192</td>
<td>14351</td>
</tr>
<tr>
<td>16384</td>
<td>7791</td>
</tr>
<tr>
<td>32768</td>
<td>4687</td>
</tr>
<tr>
<td>65536</td>
<td>1873</td>
</tr>
<tr>
<td>131072</td>
<td>1081</td>
</tr>
<tr>
<td>262144</td>
<td>681</td>
</tr>
<tr>
<td>524288</td>
<td>651</td>
</tr>
<tr>
<td>1048567</td>
<td>641</td>
</tr>
</tbody>
</table>

**Figure-5.11.1: Mobile-Agent’s Performance with Varying Chunk Size.**

![Graph showing file transfer time for varying chunk sizes](image-url)
Distributed Mining Algorithm:  
[Apriori + Mobility]

Input:
- N: Number of sites
- S: Support

Output:
- //consolidated item set for all N sites
- L: Larger Item Sets

Algorithm:

//array of item sets used to store
The item sets generated at N sites
Y[1 .. N];

i=1;
For each i <= N do
{
// Dispatching the agents to all N sites
// each identified by URL
Y[i] = send (Agent, URLi, S);
}
i=1;
For each i <= N do
{
// Consolidate the result received
from N distributed site
L= Consolidate( Y[i]);
}

When the above algorithm is used for distributed item sets, among the N number of sites, the Apriori algorithm/capability of migrated agent is locally executed at each site independently and after the successful execution of algorithm the final item set generated for that site is dispatched to the central site. Once all the item sets from all the sites are available the results are consolidated to obtain a large item set L.
Flow Diagram for Central Site Operations:

Figure-5.12: Flow Diagram for Central Site Operation in Distributed Data-Mining.

59
Flow Diagram for Remote Site Operations:

Local Large Dataset

Agents from Central Site go to Remote Sites

Execute Agents Locally at Each Remote Site: Generate Association Rules

Store: Association Rules in Local File

Dispatch Agent with Local Association Rule file to Central Site.

Figure-5.13: Flow Diagram for Remote Site Operation in Distributed Data-Mining.
From figure-5.16, it is clear that mobile-agents perform better in terms of time and network bandwidth usage compared with traditional system built over client server technology.

<table>
<thead>
<tr>
<th>File size in MB</th>
<th>Measured time in msec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File transfer + Mining at central site using traditional approach</td>
</tr>
<tr>
<td>0.001</td>
<td>782</td>
</tr>
<tr>
<td>0.002</td>
<td>820</td>
</tr>
<tr>
<td>0.003</td>
<td>823</td>
</tr>
<tr>
<td>0.1</td>
<td>875</td>
</tr>
<tr>
<td>0.2</td>
<td>887</td>
</tr>
<tr>
<td>0.3</td>
<td>818</td>
</tr>
<tr>
<td>0.4</td>
<td>830</td>
</tr>
<tr>
<td>0.5</td>
<td>887</td>
</tr>
<tr>
<td>1</td>
<td>997</td>
</tr>
<tr>
<td>2</td>
<td>1017</td>
</tr>
<tr>
<td>3</td>
<td>1030</td>
</tr>
<tr>
<td>4</td>
<td>1149</td>
</tr>
<tr>
<td>5</td>
<td>1206</td>
</tr>
<tr>
<td>10</td>
<td>2138</td>
</tr>
<tr>
<td>20</td>
<td>4979</td>
</tr>
<tr>
<td>30</td>
<td>5711</td>
</tr>
<tr>
<td>40</td>
<td>8639</td>
</tr>
<tr>
<td>50</td>
<td>10006</td>
</tr>
</tbody>
</table>

Table-1: Round Trip Time for Distributed Data-Mining Using FTP and Agents.
The percentage improvement of performance increases with decrease in the number of patterns obtained in each site because, more number of patterns, implies, more data to be carried by the mobile-agents during migration to central site. The mobile-agents are not good for data-transfer which is clear from figure-5.18 and table 3, where performance of mobile-agents are compared with File Transfer Protocol (FTP) to transfer huge data over internet. The poor performance is due to serialization and deserialization process of objects involved in migration.

![File Transfer+Mining at Central Site](image)

Figure-5.14: Traditional Approach.
However there is limitation on the capacity of the data/knowledge, which the agents carry with them, as indicated in Figure-5.17 and Table -2. Here, the agent was made to travel through several hosts and Round Trip Time (RTT) was measured.

<table>
<thead>
<tr>
<th>Agent size in MB</th>
<th>Round Trip Time in msec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Try-1</td>
</tr>
<tr>
<td>0.5</td>
<td>3585</td>
</tr>
<tr>
<td>1</td>
<td>6179</td>
</tr>
<tr>
<td>1.5</td>
<td>6530</td>
</tr>
<tr>
<td>2</td>
<td>9543</td>
</tr>
<tr>
<td>2.5</td>
<td>8742</td>
</tr>
<tr>
<td>3</td>
<td>8983</td>
</tr>
<tr>
<td>3.5</td>
<td>14150</td>
</tr>
<tr>
<td>Agent Size</td>
<td>Round Trip Time (ms)</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>4</td>
<td>13649 14080 13509 13399 13850</td>
</tr>
<tr>
<td>4.5</td>
<td>14872 13349 12939 13119 13019</td>
</tr>
<tr>
<td>5</td>
<td>13239 13720 13129 13089 12978</td>
</tr>
<tr>
<td>5.5</td>
<td>12929 13199 13098 13129 12938</td>
</tr>
<tr>
<td>6</td>
<td>13099 14621 13449 14261 13950</td>
</tr>
<tr>
<td>6.001</td>
<td>14861 13099 12958 13369 13119</td>
</tr>
<tr>
<td>6.002</td>
<td>13199 12988 13128 13199 12959</td>
</tr>
<tr>
<td>6.003</td>
<td>13489 13139 13069 13129 13219</td>
</tr>
<tr>
<td>6.004</td>
<td>13189 13259 12999 12868 13109</td>
</tr>
<tr>
<td>6.0045</td>
<td>13129 13059 13139 13259 13359</td>
</tr>
<tr>
<td>6.004 + 513 bytes</td>
<td>14050 13189 13489 13189 13359</td>
</tr>
<tr>
<td>6.004 + 514 bytes</td>
<td>13490 13189 13189 13390 13289</td>
</tr>
<tr>
<td>6.004 + 515 bytes &amp; LATER</td>
<td>Java.lang.OutOfMemoryError/Failed to Dispatch Aglet—ERROR MESSAGE</td>
</tr>
</tbody>
</table>

Table-2: Round Trip Time for Agent with Varying Agent Size.
Comparing Centralized & Decentralized Mining of data

Data Set size in Number of Records (in 1000s)

- File Transfer + Mining
- Total Time Required for Mining Remote Data & Getting Result to Central Site (in mSec)

Figure-5.16: Comparison of Centralized and Decentralized Data-Mining.
Effect on Aglet Size on Network Latency (Observed Average)

From the figure-5.17 the following observations are made:

- As the size of the agent increases, the network latency and RTT also increases minimally.
- Agent fails to dispatch at 6291455 bytes. The reason being the constraints of JVM on the size of the object.

This imposes the restriction on the size of the knowledge that the agent carries with it during migration from one host to another.
We can observe from Figure-5.11.1 and table-0 that, as the chunk size that the agent is carrying is increased, the performance of the agent for file-transfer improves. An interesting point to note is that at higher chunk size, agents perform better than FTP for file-transfer.

![FTP Vs. Aglets](image)

Figure-5.18: File Transfer Using FTP and Agent.

<table>
<thead>
<tr>
<th>Trial number</th>
<th>Using FTP</th>
<th>Using agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1150</td>
<td>6333407</td>
</tr>
<tr>
<td>02</td>
<td>1110</td>
<td>6333398</td>
</tr>
<tr>
<td>03</td>
<td>1290</td>
<td>6333418</td>
</tr>
</tbody>
</table>

Table-3: File Transfer Using FTP and Agent.
Work Conclusion:
Compared to the existing applications of similar kind, mobile-agents’ application proves to be one of the best and robust methods to handle distributed data and hence distributed data mining in faster way. This also addresses the issue of handling dynamically generated data, because the data is maintained at remote sites, which can be updated continuously, or as and when it is required. Since this approach does not require the huge amount of data transfer from remote to central site, the network resources are used optimally. Hence existing client server based distributed data-mining systems need to be rewritten using mobile-agents paradigm, to take the advantage they offer. There is a scope to develop better mechanism to consolidate the collected results (R) from different sites.

5.3. Security Issues in Mobile-Agents Applications:

Distributed system employ models in which processes are statically attached to hosts and communicate by asynchronous messages or synchronous remote procedure calls. Mobile-agent technology extends this model by including mobile processes which autonomously migrate to new host. Widespread acceptance of mobile-agents is hampered by insufficient security mechanism. The mobile-agents present new security challenges because existing security policies are inadequate. Current policies and administration practices implicitly assume a tightly coupled relationship between where the program is stored and where it is executed. Decoupling a program’s storage and execution location and allowing it to migrate during execution according to its own internal logic directly contradicts existing access-control models. This decoupling creates two entities that require protection: remote hosts and mobile-code programs that migrate to them. Still this research area is in its infancy and many approaches remain to be investigated.

Security problems fit into two categories: protecting host systems and networks from malicious agents and protecting agents from malicious hosts. There are many mechanisms like Digital Signatures and Trust Management to protect a host against
malicious agents. Several approaches tried by researchers for developing security mechanisms to combat mobile agents [7] are:

**Code Signing:** Uses authentication to identify known versions of non-malicious mobile code programs and to detect the known viruses. The main drawback of this is that it checks the known syntax patterns rather than the known or possibly malicious behaviors.

**Sandboxes:** Restrict the set of operations all program can perform on a given system. Java uses combination of sandboxing and code signing to protect hosts from possibly malicious java applets.

**Proof-carrying:** Code requires mobile-code programs to prove that they satisfy a system’s stated security policies.

The possible types of attack and value of a successful attack performed by either agents or hosts are being addressed in the paper [16] and are:

The attack that an agent may try:

- **Damage**: It destroys the target files or configuration.
- **Denial of Service**: To prevent a service or function of working module.
- **Breach of privacy**: Undesired access, removal of data from a system or an agent.
- **Harassment**: Annoy people by showing unwanted pictures or typing to screen with an interval that causes flickering at frequency which causes seizures in sensitive people.
- **Social Engineering**: Includes manipulation of people, hosts or agents by using misinformation. Ex: play a role of administrator and request user's passwords.

The attack of host against agents can be:

- **Spying out code**: The host can try to solve out the functioning of an agent due to the exposure of code.
- **Spying out data**: If the host gets access to the private data of an agent, the consequences can be very severe.
- **Incorrect execution of code**: An attacking host may change the way it executes the code of an agent.
Masquerading of the Host: During an agent’s transmission a malicious host can intercept or copy and manipulate and then transfer to other host.

Denial of execution: The host may just ignore the agent and not execute it.

Spying out Interaction with Other Agents: A malicious host can get information about the private data stored in the system. It can also claim to be the agent to other agents or mask itself as a partner of the agent.

Returning wrong results of the System-calls issued by the Agent: System-calls are the calls that only the host system can handle. If the host returns falsified answers to the calls, it could get a number of benefits including access to the agent’s private data by masquerading to be the agent’s host.

Proposed Model:

Hence there is a need for every host to authenticate the agent, its owner, assigns the resource limits. An attack of a host is considered a very difficult problem [16]. Some authors have even declared the problem not solvable. In the proposed model, it is assumed that the Host must have access to the instruction it is running and it will not harm the agent-code.

The proposed framework is termed as: “Hierarchical Domain Based Authentication Framework for the Mobile-Agents Application (HDBA-MA)” and is as detailed below:
HDBA-MA is based on the protocol used in many real systems- "KERBEROSE", Public Key cryptography and Message Digest. The heart of HDBA-MA is the Authentication Centers (AC) and is similar to Public Key servers and Digital Certificate issuing authority. The ACs are responsible for:

1. Maintaining the registration Information for each host.
2. Provide the individual secure key (Ks) for each host registered in it, for establishing communication in either direction.
3. Issuing the Personal Identification Number (PIN) for each agent for the host registered in it.
4. Validate the identity and the integrity of the agent when it is migrated to the other host.

ACs are organized in the form of hierarchies, to accommodate more hosts participating in several business domains across the world in future. Conceptually the AC's are organized in the form of the tree. To create a new AC, permission is required of the AC’s under which it will be included.

The tree structure for ACs:

```
AICTE

KA  MH

KUD  VTU  SHI  PUN
```

Figure-5.19: Tree Structure of Authentication Center (AC).
There are five phases in HDBA-MA: and are as described below:

I. Registration Phase:

All the hosts, which are responsible for creating the agents, that will migrate and participate in business process at some other host, need to register themselves with the nearest AC in its area. It needs to supply the following information about it, to AC for registration process:

- Address in the form of IP Address: port number, where the context for the mobile-agent is running
- Its Public Key

II. Personal Identification Phase (PIN):

The Host ‘X’ sends the intention $K_{s}[X^{*}, t^{*}, a^{*}, MD, Dest^{*}]$ of sending the agent (to destination hosts) to its AC, which includes its details $X^{*}$, time stamp $t^{*}$, agents details $a^{*}$, Message Digest ‘MD’ and the list of Hosts to which agent ‘a’ wants to visit. All these are encrypted using the secret key $K_{s}$. Since $K_{s}$ is known to only ‘X’ and its AC, no one can forge this message or send false message for the PIN. Then AC prepares the PIN, which includes: its details $AC^{*}$, its public key $PK(AC)$, Destination-List along with it’s Public keys in pair $PKD^{*}$ and the encrypted ID $PK(AC)[a^{*}, t^{*}, Ra]$, where Ra is the random number. AC stores this PIN along with X’s initial request. The PIN is encrypted using the secret key $K_{s}$ and sent to X: i.e. $K_{s}[PIN]$.

III. Agent Migration Phase:

The host ‘X’ prepares the agent a $[PIN, a.PK_{s}]$. Then the host ‘X’ sends the agent to the first host of the Destination-List using a Public Key of the first destination host:

$PK_{i=1} [agent ‘a’ + Hash Algorithm (HA) used to generate Message Digest (MD)]$
Subsequently, the Knowledge of Initial Visit Plan in the form of pairs i.e., Host Address and its Public Key \[ \{ \text{Dest*}, \text{PK*} \} \] encrypted using the agents 'a' public key is sent to 'a'.

IV. Verification Phase: Authority, Authentication and Integrity of Agent.

When the agent 'a' migrates to the first host 'Y' in its Destination-List, at 'Y' the following activities takes place:

1. \( R = \{ \text{agent 'a' } + \text{HA} \} \)
2. If 'a' is permitted to enter 'Y' the got step 3 else exit
3. \( \text{MD} = \text{HA}(R) \)
4. \( \text{PIN} = \text{a.PIN} \)
5. \( \text{AC} = \text{a.AC} \)
6. \( \text{PK} = \text{PIN.PK(AC)} \)
7. Send request \( \text{PK (AC) [PKy, PIN, MD]} \) to AC to authenticate 'a' and also to check Integrity of the agent i.e., 'R'. If 'not ok' exit else proceed.

V. Business and Migration Phase:

Execute 'a' in the context of 'Y' with whatever authorization it has. Agent 'a' creates the knowledge say 'K' during its execution in 'Y' which also includes the variations in the visit-plan along with their Public Key in coordination with 'Y'. Agent 'a' when it completes its task in 'Y' sends R to next destination available in its knowledge (Destination-List) with the appropriate Public key. When 'R' reaches next destination, this agent 'a' sends the knowledge 'K', using its Public Key i.e. a.PK [K] and destroys itself after confirmation of receipt of knowledge 'K' by 'R' at next destination.

For illustration, consider the host (X) which has registered (210.212.5.15:4048: x-public-key :) in the AC named KUD wants to do business by sending an agent, say 'a' to the host (Y) which is registered (190.190.5.12:5000: y-public-key :) in the AC named 'SHI' and to the host (Z) which is registered (190.195.8.19:6000: z-public-key :) in the AC...
named 'PUN'. The figure-5.20 shows the working of the proposed protocol for authentication.

Work Conclusion:

Security model for mobile-agents – "Hierarchical Domain Based Authentication Framework for Mobile-Agents Application" comes at the expense of time and resources at the context in which agents run at each host. There is a requirement to integrate this proposed framework in to the context and establishment of standardization for inter context communication across different vendors proposing mobile-agents.
KS[X*, t, a*, MD, Dest*]

KS[PIN]

PK_{i*} [ a[PIN, a.PKs] + (HA) ]

a.PK[Knowledge = {Dest*, PK*}]

PK (AC) [PKy, PIN, MD]

PKy[Yes/No]

R = a[PIN, a.PKs] + (HA)

'a' does Business in 'Y' & Creates Knowledge - K
Figure-5.20: Sequence Diagram Showing the Working of Proposed Agent’s Authentication Protocol.
5.4. Agent Tracking:

Distributed system employs models in which processes are statically attached to hosts and communicate by asynchronous messages or synchronous remote procedure calls. Mobile-agent technology extends this model by including mobile processes, which autonomously migrate to new hosts. A mobile-agent is a running program that can move from host to host in a network whenever and wherever it chooses. There are many applications of mobile-agents like Network Monitoring, Resource Discovery and Allocations in Grid Computing, Network Routing, etc. Interestingly, E-market opens up an exciting world for software agents and middle agents—a market place, where automated agents can represent humans and sometimes even outperform them. Mobile-agents are also used to provide context aware services in ubiquitous applications. In a distributed system, failure can occur in any hardware and software components. For any system, reliable operation is attained when all components of the system work according to specification. Mobile-agent can get lost due to host failure or many network problems. Therefore reliability is a vital issue for deploying the mobile-agent system.

Extensive research has been conducted in the areas of survivability and fault tolerance. Stefan Pleisch and Andre Schiper [36] adopt the use of replication and masking, employing replicated servers to mask failures. Manfred Dalmeijer and his colleagues [37] use checkpoint manager to monitor all agents. This manager is responsible for tracking all agents and restarting those that have failed. Taha Osman, Waleed Wagealla, and Andrzej Bargiela [38] analyze an execution model for agent platforms to develop a pragmatic framework for fault tolerance in agent systems. This framework deploys a communication pair, independent check pointing strategy. Simon Pears, Jie Xu, and Cornelia Boldyreff [39] use two exception handling approaches operating on different servers to maintain mobile agent’s availability. Luis Moura Silva, Vitor Batista, and Joao Gabriel Silva [40] present a set of fault tolerant techniques, including fault detection, check pointing and restart, software rebuilding, and reconfigurable itinerary.
Michael R Lyu, Xinyu Chen, and Tsz Yeung Wong [35] uses the approach of Dag Johansen and his colleagues, which employs three types of agents to detect servers and agent failures and recover services in mobile-agent systems. An actual agent is a common mobile-agent that performs specific computations for its owner. Witness agents monitor the actual agent and detect whenever it is lost. A probe location recovers the failed actual agent and the witness agents. A peer-to-peer message-passing mechanism stands between each actual agent and its witness agents to perform failure detection and recovery through time-bound information exchange, a log records the actual agent’s action. When failures occur, the system performs rollback recovery to abort uncommitted actions. They use check pointed data to recover the lost actual agent.

**Proposed Model:**

The previous section addresses the various solutions for the fault tolerance but doesn’t address the agent tracking from the perspective of the owner of the agent like: where is the agent? and what is its status? Most real world agent applications need some level of agent tracking in order to ensure error free operations. Agent systems can be hard to debug due to their distributed asynchronous nature. Hence special attention needs to be given to diagnostic tools for agent-based systems before they are deployed in the real world.

There are some obvious possibilities of agent tracking from owner’s perspective and they are:

1. **Arrival-Migrate Logging (AML):** Here, immediately after the agent arrives at each remote host, it registers its details and also leaves the future travel plan before migrating. This in turn will be communicated to owner of the agent by the special designated agent of each context through message. The disadvantage of this approach is the use of resources for maintaining agents’ details at each remote host.

2. **Remote Searching:** Here many traditional searching techniques can be thought where, agent will visit every host in the network to find the ‘agent’, the owner is looking for. This does not seem to be effective and efficient in terms of network scalability as agent may also change its travel plan during the migration.
Here we are proposing the Agent Tracking (AT), which needs to be integrated with the available agent context, instead of leaving it to the level of application development. The 'AT' uses the following concepts:

1. Every host has a specially designated agent/service called TRACKER. (To be integrated with context. Now replication concept used to make tracker free from failure). The responsibility of the tracker is to maintain all the details of the agents that are created in its environment/context and also about the agents that migrate from this host. Once the agent registers with TRACKER, the tracker irrespective of agent's location regularly polls the agents and maintains their status. If the communication fails due to some reasons, the TRACKER records status for that agent as 'not available' thus takes care of the messages, which have not been successfully delivered to the tracker. It allows every agent to send to it, the following types of messages:

- Reg:
- Unreg:
- Find:
- Forward:

Every time the agent arrives at a new remote host, it sends the message of type 'Reg:' to the TRACKER, telling its current location details. The message type 'Unreg:' will remove the details of the agent that has sent this message from the database of the TRACKER. The message type 'Find:' will return the caller, the proxy i.e. the reference of the agent, which the caller is interested in. Using this reference the caller may collaborate with the agent for some transaction. Immediately after instantiating, every agent say 'A', will register itself with the TRACKER in its owner's context/environment and also subsequently at arriving at each remote host, by sending a message of type 'Reg:' If the message of type 'Reg' fails to get recorded at owner's TRACKER due to communication failure i.e. agent 'A' at remote host fails to receive the acknowledgement, it then repeats the process of registration at regular intervals of time. During this process, if 'A' wishes to migrate to another host, then instead of waiting there, it sends its current
details to the TRACKER of the current host by sending message of type ‘Forward’ and after confirmation, ‘A’ migrates. The TRACKER of the current host in turn communicates, at later stage to A’s owner TRACKER. Thus successful delivery of the message containing status of every agent is achieved.

2. Every host will send a message of type ‘Unreg:’ to TRACKER at its owner’s context/environment, to indicate the end of its assigned responsibilities. Once this happens, no one will be able to collaborate or know its status in future, as that agent no longer exists.

3. Every message of the type “Forward”, received by the TRACKER, will be forwarded to respective destination.

The above concept was tested using IBM’s Tahiti Server (AGLET) with Java programming environment and the application of resource identification at each node in a network. The system developed found useful as proposed, with the assumption that, the host and all the agents are free from failures.

The ‘AT’ does consume little more resources at every host and is scalable. This is because of additional process ‘TRACKER’ running at every host as part of agent’s context. All agents maintain their current location plan at their owner’s place/context. It allows locating any agent and collaborates with it for any transaction from owner’s perspective, thus relieving the application programmer from the responsibility of tracking the agents. The same model can be extended to make TRACKER aware of the details of all the other agents doing transactions at any remote host.

The detailed working of ‘AT’ is shown in the figure- 5.21. Sequence diagram-1 (Fig-5.21) shows how agent ‘A’ gets created, registers with TRACKER, migrates to remote host and registers subsequently after arriving at every new location. Sequence diagram-2 (Fig-5.21) shows how owner of agent i.e. caller can make use of the reference of the agent ‘A’ which it has created and sent to different sites for doing transactions and also how agent unregisters itself with TRACKER before completing its responsibilities.

Finally, Agent-tracking allows locating any agent and collaborates with it for any transactions from owner’s perspective, thus relieving the application programmer the
responsibility of tracking the agents. The same mechanism can be extended to make TRACKER aware of the details of all other agents doing business at any remote host and replication strategy can be used to make TRACKER, free from failures. However, this protocol comes at the expense of time and resources at the context in which agents run at each host. There is requirement to integrate this framework in to the context.

This concept was published in international conference on computational intelligence for modelling, control and automation- CIMCA-2005 and is published by IEEE Press in USA.

Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) – Volume-1
Pages: 752 - 756
Year of Publication: 2005
ISBN: 0-7695-2504-0-01
Authors U. P. Kulkami SDM Engineering College, Dharwad- INDIA
A R. Yardi Walchand College of Engineering, Sangli- INDIA
Publisher IEEE Computer Society Washington, DC, USA
Sequence Diagram -1

CREATE AGENT "A"

A-SEND 'Reg:'

Ack

A-DO BUSINESS IN OWNERS CONTEXT

A-MIGRATE TO REMOTE HOST

A-SEND 'Reg' to its OWNER. On failure send 'Forward to Local Tracker'

Polling 'A' at regular intervals

At regular large intervals

On failure

On behalf of 'A' send 'Reg'

At Later Stage

RECORD A's DETAILS

A-DO BUSINESS IN REMOTE CONTEXT

MIGRATE TO NEXT HOST
Figure-5.21: Proposed Agent Tracking Protocol.