Agile Systems Manager for Enterprise Wireless Networks

Sandeep Adwankar and Venu Vasudevan

Networks and Infrastructure Department,
Motorola Labs,
1301, E. Algonquin Road,
IL02-2240, Schaumburg, IL 60196,
sandeep.adwankar@motorola.com, venuv@labs.mot.com

Abstract. Advances in enterprise wireless networks with wireless LAN along with the need for a pervasive computing is leading to a very large growth in a number of managed elements. The conventional management model of a centralized manager does not work well in this highly evolving network that is characterized by mobility and intermittent connectivity. This paper motivates a need for an agile systems manager, one that is resource and network aware and redistributes the management logic within a ìvirtualî management station to operate effectively in changing network conditions. This paper proposes mobile agent based approach for systems management infrastructure that can deploy, reconfigure, load balance and react to environment change autonomously. The paper presents a performance management implementation for managing a mid-sized network composed of combination of wireless and wired network elements. The paper presents experimental results that show significant advantages of the agile management system over the conventional management system.

1 Introduction

With emergence of a wireless LAN, distinction between "fixed" (wired) and mobile wireless enterprise is blurring, which has implications on systems management. System management approaches for wired networks have relied on the fact that they have a relatively fixed set of network elements (and topology), connected together with high-bandwidth links. The wireless enterprise has existed in a different universe of protocols, hardware and management mechanisms. Now, with the ability of mobile laptop or handheld to enter an enterprise and "plug-in" to (and disconnect from) the fixed network, the network has a continually changing element set and topology, and a mixed breed of network links with different bandwidth capabilities. Over time, the wireless subset of this network is likely to become a significant fraction even in fixed enterprises, due to the difficulty in installing and expanding wired networks to accommodate personnel and organizational changes in an enterprise.

Additionally, the mobile workplace and associated mobile devices places new challenges on systems management such as coordination of software upgrades across

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-45812-8_28

K.C. Almeroth and M. Hasan (Eds.): MMNS 2002, LNCS 2496, pp. 62-76, 2002.

both static and mobile devices, load balancing and performance monitoring of dynamic network architectures.

Managing a dynamic network with conventional centralized management model will become progressively more cumbersome, and motivates the need for agile management architecture, where the management system re-organizes based on the changes in network. This paper explores elements of agile management, and proposes an agent based management architecture that exhibits such agility. The following attributes characterize an *agile* systems manager (ASM) :

- **Dynamic and Distributed**

An agile systems manager has management logic distributed throughout the network and location of manager components can be changed. The definition of manager includes all devices from which one can retrieve enterprise systems data and network data and can perform management operation. NEW management functionality can be incrementally added with self-distributing components.

- **Decentralized and Fault-tolerant**

The architecture is inherently decentralized, as there is no centralized management station where all intelligence is located and hence there is no central point of failure. This is particularly important for wireless networks as both managed elements and manager components can be intermittently connected. The architecture does not assume that all management agents are dumb and perform only get or set. The management functionality is distributed as autonomous components having their own fault recovery process.

- **Resource-Aware**

The agile systems manager is network and resource aware. It positions the management components according to dynamic network characteristics. The network-aware components are placed closer to managed hosts to conserve bandwidth of most frequent management operations. The management components requiring more CPU resources are placed on hosts having spare CPU computing cycles.

- **Mobile and Adaptive**

Agile management components have built-in mobility with operator-configurable rules that enable them to find the best location to migrate to for optimal operation. Manager composition and topology continually responds to a continually updated, dynamically discovered network topology. Reactive mobility allows an agile systems manager to survive attacks without the need for centralized, manual operator procedures.

- **Scalable**

The conventional management system micro manages due to central intelligence, thus it is not very scalable. The agile systems manager being decentralized; is able to scale and move to different administrative domains and send only useful and critical data to network operators. At the same time, operator has observability to all elements of the agile systems manager.

This paper presents our experiences in build an Agile Systems Management platform, whose capabilities are leveraged in an adaptive performance manager. The capabilities of the management platform are equally relevant to configuration and fault management problems. The rest of the paper is organized as follows. Section 2

introduces systems architecture of ASM. Section 3 gives overview of the ASM infrastructure and its capabilities of dynamic configurability and self-adaptation . Section 4 describes a network performance management application that leverages the platforms self-configuration capabilities. Section 5 describes discovery phase in the ASM and provides experimental results that shows more than 40% improvement in discovery operation due to mobility. Section 6 describes dynamic characteristics of performance monitoring with results. Section 7 discusses fault tolerance aspects and experimental results showing resilience of ASM under denial of service attack. Section 8 presents related work and Section 9 concludes with directions of future work.

2 Architecture of Agile Systems Manager

In a conventional systems manager, all management functions are centralized into manager platform. In ASM, management functions are distributed in components that have infrastructure substrate as a base. All components are autonomous and do not depend on centralized intelligence to perform management operations. The primary components of ASM are shown in Figure 1.

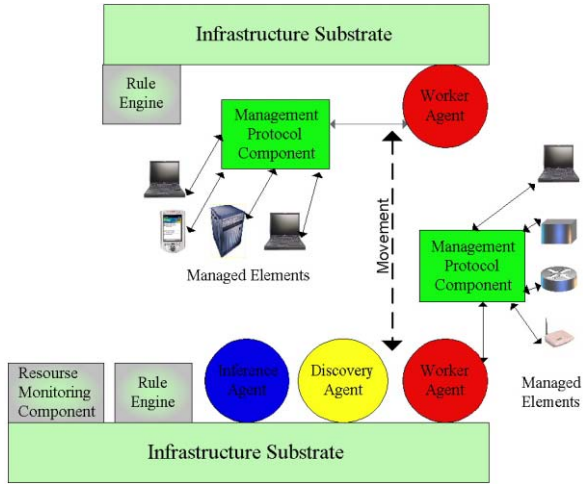

Fig. 1. Agile Systems Manager Architecture

2.1 Worker Agent

The worker agent is a primary management activity component of ASM. Worker agents are distributed throughout the network and they use management protocol components to invoke management operations on managed elements. Worker agents communicate with other agents using Tuple Space based protocol supported by the

infrastructure substrate. The store-and-forward nature of tuple spaces allows communicating agents to reside on intermittently connected devices.

A worker agent carries a rule certificate, which encapsulates agent-specific policies for optimal location, and migration and cloning criteria. R rule certificates provide application reconfiguration policies to be defined and enforced in distributed fashion on a per agent basis.

The Performance Monitoring (PM) agent is a type of worker agent implemented to monitor performance of a set of managed elements. A graphical interface is associated with PM agents for debugging purposes. This allows the PM application developer to visually track agent movement, and potentially tune their rule certificates to optimize migration patterns. A snapshot of PM agent monitoring four hosts in 145.1.80 subnet is shown in Figure 2. This PM agent monitors ipInReceives, total number of input datagrams received from interfaces, including those in received in error and icmpInEchos, number of ICMP echo messages received.

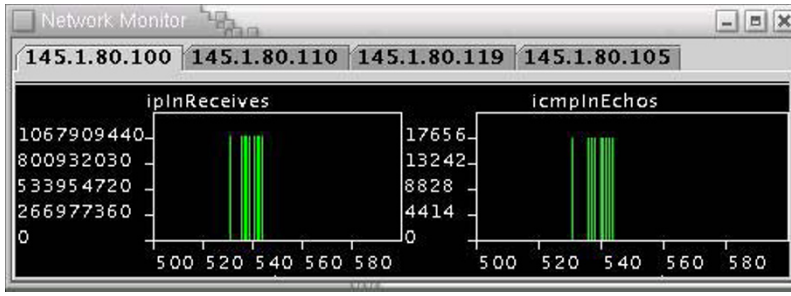

Fig. 2. PM Agent monitoring input packets and ICMP packets of 4 hosts in 145.1.80 network

2.2 Discovery Agent

The discovery agent dynamically discovers all active and manageable elements in the assigned domain by continuously monitoring the network. It provides this information to other agents in the system like worker agents and inference agent. The discovery agent maintains a list of active elements and manageable elements. It continuously monitors the network to check if any element becomes active and if so, it puts it in the active list. The discovery agent also monitors if any active element becomes manageable (due to start of management agent) and if so, puts it in the manageable element list.

In case of discovery system, our approach was not to come up with clever discovery algorithm, but to come up with mechanism that can perform multiple localized efficient discovery operations in terms of time taken for discovery. Discovering a manageable element is two-step process:

1. Finding whether the element is active. The discovery agent has a ping like implementation, in which a socket connection is tried with the element. If there is no route to host or if the host is unknown, that element is considered inactive.

2. Finding presence of SNMP agent on an element. The discovery agent creates a SNMP session, establishes the active element as peer and performs SNMP get operation on that active element. If this operation is successful, it identifies a working SNMP agent on that element with appropriate MIB.

2.3 Inference Agent

The inference agent serves both as an entity that makes high-level inferences from collected telemetry, and as the console for network operators to control and observe deployed network management applications. It interacts with worker and discovery agents to maintain (and log) an accurate view of network topology and state. Figure 3 shows the snapshot of an inference agent showing an agent-based view of the managed system. The *Agent View* panel on Figure 3 shows a mapping between agents and the hosts they manage. The *Monitor Data* panel shows telemetry on a per host basis. This telemetry is information that is processed (aggregated and averaged) by PM agents.

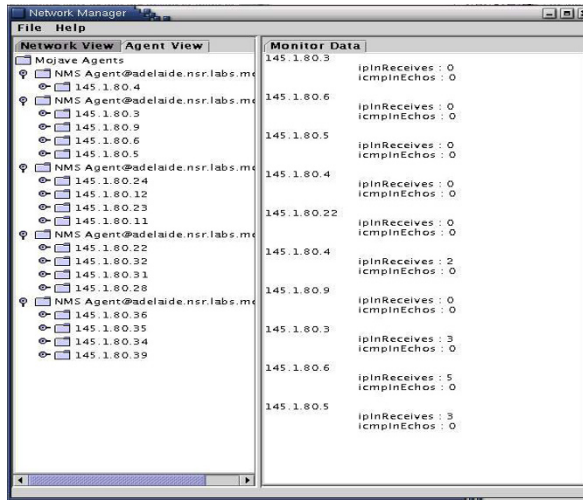

Fig. 3. Inference Agent showing Agent view and monitor data

The Inference agent can make use of monitor data to infer things such as top 10 nodes generating maximum network traffic or top 10 nodes under denial of service attack.

2.4 Management Protocol Component

Worker agents use a management protocol component (e.g. SNMP, CMIP or IIOP) to execute management operations on managed elements. This component allows other agents in the system to be agnostic about the underlying physical management protocol.

3 Infrastructure Substrate in ASM

The infrastructure substrate provides a number of support services to enable decentralized resource-aware, network-aware adaptation. To achieve this, it relies on the capabilities provided by the Mojave mobile agent system [2] [3]. Mojave supports a two-layer model of agents: a functional part and a policy (rule certificate) part. This model is key to enabling customizable adaptation policies, which can be enforced in a decentralized manner. The rest of this section describes the various components of Mojave and the role they play in architecting agile, auto-configuring systems management applications.

3.1 Pod

The Pod is an agent execution engine of the ASM. It provides runtime context and support services for the agent execution. To every incoming agent, pod provides a thread of execution, access to rule engine and resource monitoring component. Each pod is implemented as a Jini[5] service and is registered with a lookup service, thus each pod is able to find presence of other pods. Agents query host pod (i.e. pod where they are currently executing) to determine the places (pods) where they can move.

3.2 Resource Monitoring Component

The resource monitoring component makes ASM network and environment aware. Worker components make mobility or clone decisions based on resources available on a network and a host. The resource information such as bandwidth, delay, CPU spare cycles, available memory, network topology is gathered by a resource monitoring component and is made available to a number of worker agents. A resource monitoring component is associated with rule certificates of different worker agents.

3.3 Rule Engine

A rule engine on a pod tracks rule certificates of all agents in the pod. The XML based rule certificate consists of a set of event-action rules, with the event describing environmental conditions that can cause an agent to take some action, and the action clause describing what the agent needs to do. The rule engine maintains relationship with rules certificates, resource monitoring components, and agents. The rule engine is responsible for parsing rules, adding rules to the engine, firing rules and triggering agents.

3.4 Communication Mechanism

A pod needs communication infrastructure to transport an agent from one pod to another. Agents need a way to communicate with another agent or group of agents. A tuplespace satisfies this network middleware requirement by providing capability of network communication buffer. The tuplespace is a globally shared, associatively

addressed memory space that is organized as a bag of tuples. A tuple is a vector of typed values or fields. An agent can register with a tuple space with a specific tuple template. It will then receive callback, if another agent writes tuple of same template to the tuple space, allowing it to communicate with group of agents. Pods use same mechanism to transport agents. They register with the tuple space with a template having a serialized agent as one of the field. When any pod writes serialized agent to the tuple space, it will receive callback of serialized agent. The Liaison is an entity that implements ASMís interface to the tuple space. IBM TSpaces [9] was used as a tuple space implementation.

3.5 Lookup and Leasing Mechanism

Pods and agents need mechanism to find other pods in the system. It also needs mechanism to remove outdated references for non existent pods or agents that did not or could not clean up. Jini provides architecture and implementation of lookup and leasing mechanism. So when agent service is plugged into a network of Jini services, it advertises itself by publishing an AgentProxy, a Java object that implements the agent functionality. The Pod finds this agent by looking for an object that matches agent attributes. When it gets the serviceís published object, it downloads agent code it needs and hands it over to an agent manager in a pod to start the agent in a new thread.

4 Network Performance Management

The network performance management application is built to show one example of agile systems manager that demonstrates self-installation, self-reconfiguration and load-distribution aspects in real world environment. The managed network for performance management is an array of subnetworks each having around 255 hosts connected by Ethernet and wireless LAN.

The overall performance monitoring (PM) task is to measure the network stress. The network stress is an operator configurable function and is defined as a function of different network parameters like IP packet traffic, ICMP in/out, TCP/UDP connections for hosts, bandwidth and latency. This performance monitoring task is performed for a network from a ìmanagement stationî which itself is a collection of leased computers. Thus the management application runs on machines that are not dedicated to this management task. Therefore, the PM task has to share cycles with either other systems management tasks, or in the case where PM task is running on regular user machines, with non systems management tasks. The management task runs over some subset of the 255 machines on which pods are installed.

The PM task is divided into a group of identical PM agents, which are type of worker agents. Each PM agent monitors subset of IP addresses in the 255 node subnetwork. The network operator responsible for this system, instantiates the inference agent and provides list of subnetworks to be monitored. The network operator also customizes XML rule certificate for PM agent. The agent rule certificate contains rules for

- **Cloning**

The clone rule certificate lists maximum number of managed elements a single PM agent can monitor before it needs to take action to create clone for itself.

- **Mobility**

If load on a host pod is more than that specified in the loading function (combination of parameters like number of processes, CPU spare cycles, available memory etc.), then the agent tries to find a pod with lesser loading function and move there. Thus, agent can move to host having higher computational availability.

The rule certificate for mobility to protect agent under denial of service attack is shown below:

```
<watchFor action=ímoveí>
   <podState>
        <sentry name = ì ICMPSentryî  op=î GREATERî  value=î 40î  value-
        Type=î Integerî >
                com.motorola.mojave.snmp.sentry.ICMPSentry
        </sentry>
   </podState>
</watchFor>
```

This rule certificate relies on ICMP [10] resource monitoring component. With this rule, if ICMP packets for a pod are more than specified threshold (In this 40) then assume it is a denial of service attack, so move agent to another pod. Incoming agent will hand over this rule certificate to pod. The rule engine associates a resource monitoring component (ICMPSentry) monitoring ICMP packets with this rule certificate. The ICMPsentry will periodically sample ICMP packet count and if the sample value exceeds threshold, it will notify the rule engine. The rule engine performs dependency checking of this rule with all other rules contained in the rules certificate. If that leads to firing of the rule then action corresponding to that rule i.e. moving agent to another pod is executed. The rule Engine performs this check for all agents having rules about ICMP packet count.

5 Managed Element Discovery in ASM

The element discovery process is common for all types of network management activities. The discovery process is a continual process for detecting presence of any new managed element. The discovery process is usually first to start in the ASM and it requires setup of the ASM as shown in Figure 4. The steps for initial startup of agents are described below. The network operator starts infrastructure substrate that starts Jini lookup service, tuple space server and pods.

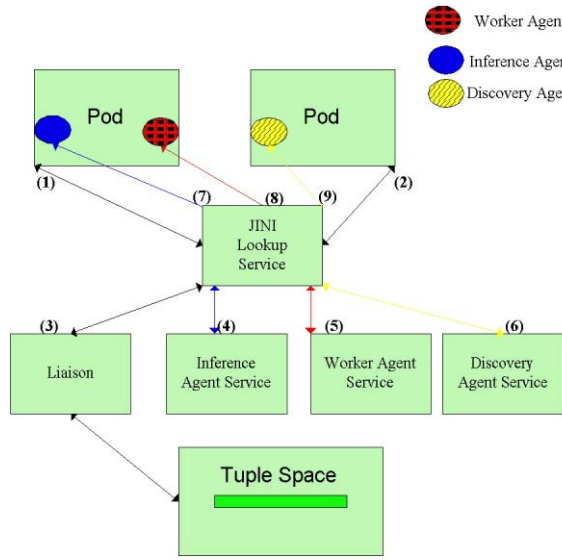

Fig. 4. Lookup Mechanism in JINI based Infrastructure Substrate

Steps 1,2: Pods register with Jini lookup service with certain name as attribute.
Step 3: The liaison which acts as wrapper for tuple space registers with the lookup service.
Steps 4,5,6: The network operator starts Jini agent services for all types of agents. These agent services register with lookup service and are responsible for starting as many agents of that type.
Step 7: The network operator starts inference agent on a pod.
Steps 8,9: The network operator starts PM agent and discovery agent in the pod.

PM agents and discovery agents establish a path of communication by registering for event channels with liaison (tuple space). A discovery agent is created on a pod that may not be on the subnetwork to be managed. The discovery agent queries host pod for presence of pods on the specified subnet. If a pod is found, discovery agent moves to the specified subnet. This move operation significantly reduces the subnet discovery time as shown in Figure 5. This figure shows the comparison of time taken when discovery agent discovers managed elements in subnet 145.1.80 while executing on another subnet 173.23.93 compared to time taken when it is moved and executed from a pod on 145.1.80 subnet. Thus there is improvement of more than 40% in the same discovery operation time due to mobility.

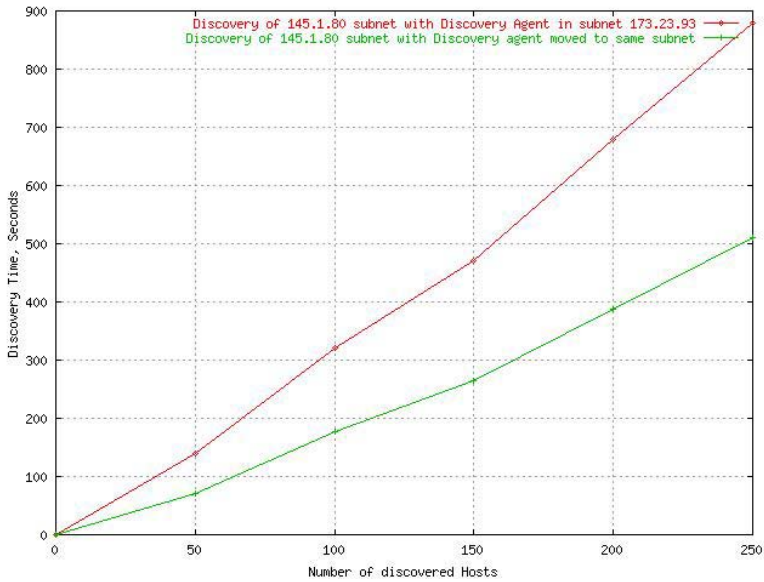

Fig. 5. Comparison of discovery time for discovery agent in same network with when it moved to different subnet

6 Performance Monitoring in ASM

The performance monitoring of a network in the ASM is accomplished by a set of distributed PM agents. These PM agents are created and positioned dynamically depending on a network topology, presence of infrastructure substrate in the monitored network and a rule certificate of a PM agent. The network operator starts a PM agent (Step 7 in Figure 4) and creation and positioning of new PM agents is achieved using cloning and movement of this agent. Figure 6 shows these clone and move phases in the ASM. The inference agent is not shown in the figure to reduce complexity, but it is listening to events from the discovery agent.

The clone, move operations steps as shown in Figure 6 are described below.

1. The discovery agent discovers presence of new active, manageable host.

2. The discovery agent writes the IP address of that host in the tuple space.

3. The PM agent is registered for listening to such tuple template. There will be eventRegister callback on the PM agent, as tuple of that template is written to the tuple space. The PM agent receives this tuple as part of callback and obtains IP address of newly discovered host. The clone rule for PM agents is ì If number of monitored hosts for an agent exceeds specified limit, clone the agentî . Thus, if the specified limit is four, then maximum number of hosts that PM agent can monitor is four. If clone rule is not triggered, then the PM agent monitors this host. It opens a new SNMP session, establishes the new host IP as peer and starts periodic SNMP ì getî for performance variables.

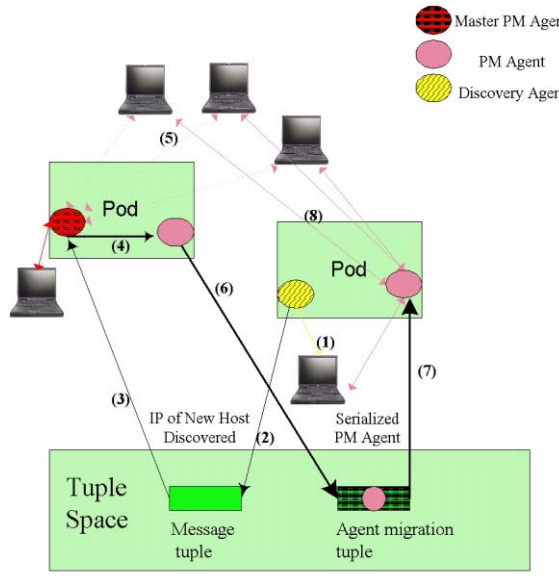

Fig. 6. Clone and Mobility in ASM

4. The discovery agent continues to send discovery events with new IP addresses and if number of hosts being monitored exceeds specified limit (say four), then PM agent clones and becomes master and creates a cloned PM agent and assigns a disjoint set of IP addresses (maximum of four, in this case) to the cloned PM agent.

5. The cloned PM agent becomes slave agent responsible for those set of IP addresses. This clone operation causes an event to be sent to the inference agent and it accordingly updates its internal data structures and correspondingly the operator GUI.

6. The cloned PM agent uses distance metric resource monitoring component (described in next subsection) to locate itself near hosts it is managing. The cloned PM agent will make call to the host pod to move itself to the new pod. The host pod will serialize the agent and will write the serialized agent in the tuple space.

7. The tuple space will invoke callback on registered pods, the new pod will take serialized PM agent tuple and will reconstruct agent object, allocate a thread of execution.

8. The cloned PM agent creates SNMP session for assigned hosts and starts monitoring those hosts.

6.1 Distance Metric Resource Monitoring Component

Distance metric resource monitoring component is used to determine the optimal location of PM agents. Since a PM agent is monitoring performance of set of hosts, it is best located at a location near to all or majority of hosts.

PM agents can execute only on agent execution engines i.e. Pods. The PM agent's task is to monitor a given set of hosts. To minimize latency in monitoring, PM agent

should be moved to a pod that is closest in terms of time taken to monitor those set of hosts. The distance metric component finds such a pod. It finds this pod by following way: The distance metric component determines presence of pods from Jini lookup service. From each of these pods it initiates connection to set of hosts and gathers latency information. The pod that gives the least value for latency is the best pod to move to.

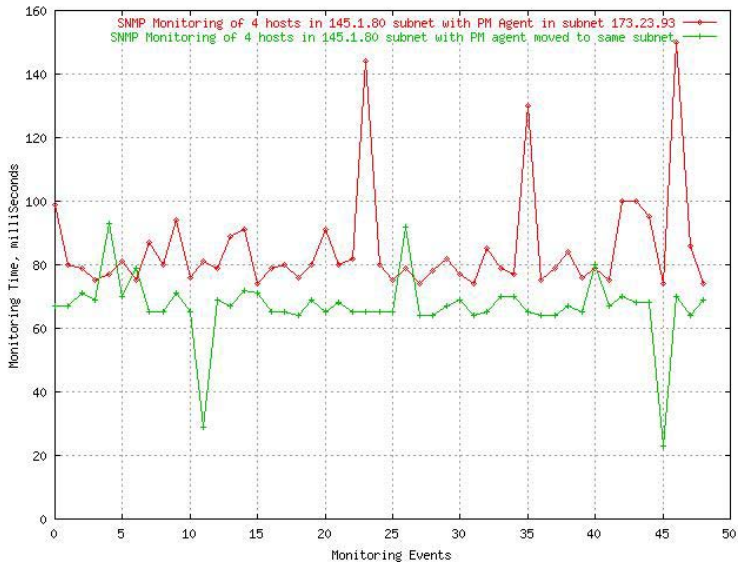

Fig. 7. Comparison of monitoring time of PM agent in different network with when it moved to same network

The Figure 7 shows the benefit of mobility of the PM agent. The PM agent using distance metric resource monitoring component moves from a different subnet to subnet on which managed host are present. There is improvement of more than 15% because of this mobility. In the case of PM agents, the increase in benefit is smaller compared to that observed for discovery agent. This is because the task accomplished by PM agent is significantly less in complexity compared to that of the discovery agent.

7 Observations of ASM

ASM based performance management is performed on two subnetworks consisting of both wired and wireless nodes over number of days. It was observed that the ASM is considerable stable with regular plug in and out of wireless LAN nodes.

7.1 Fault Tolerance and Load Balancing

The master PM agent has responsibility for recreating each of the failed cloned agents. The inference agent is responsible for recreating each of the failed master PM agent. The mechanism used for making inference agent fail-safe is by providing standard passive replication with the help of a backup inference agent.

The ASM load balances itself with the help of the mobility rules in PM agent certificates. If the operator needs to monitor the performance of four IP subnets, PM and discovery agents are dispatched to each of the four subnets and they perform the local processing by looking at large number of network parameters. These distributed PM agents send only parameters that exceed the operator specified threshold to the operatorís management console leading to a scalable solution.

7.2 Experimental Observations of Agent Operations

The Table 1 shows time taken by agents in operations such as creation, movement and cloning [15]. The last two observations show the survivability aspects of ASM. The denial of service attack is launched on host by sending very large number of ICMP packets using flood ping. For three PM agents it took ten seconds to stop their execution on machine under attack, find safe machine with a pod, which satisfies load function, move to that pod and resuming operation from that safe machine. It took 20 seconds to move 11 PM agents to safe pods on the network.

Table 1. Performance Analysis of ASM

Agent Operations	Time in Seconds
PM, Inference agent creation at a remote place	4
Discovery agent creation at a remote place	1
PM agent movement between two different hosts	4
PM agent clone operation on a same host	1
PM agent traversal across 11 hosts (10 Moves)	44
Movement of 3 PM agents simultaneously in response to the denial of service (DOS) attack on the host.	10
Movement of 11 PM agents in response to the DOS.	20

7.3 Security

The security concerns is mobile agents technology are pertinent to mobile agent implementation and scope. In our mobile agent implementation, we leveraged J2SE security manager. Each agent execution engine has comprehensive Java policy file, which governs the set of operations that agent, can perform. Since we scope agile systems manager for "enterprise networks", agents and pods are assumed to have trusted relationship. Thus any agent created by a network operator is able to execute on any pod. Mojave should be able to leverage the security improvements in Jini (e.g. Davis) as they are made available to the community at large.

8 Related Work

Goldszmidt et al. [1] describes an approach of distributed management with help of programs that can be dispatched to remote processes and can invoke delegation procedure stored. However, unlike in the ASM, the delegation procedures themselves are not downloadable. IBMís NetScript [12] adopts a script based approach for providing mobility in systems management, with HTTP as the agent transport. Netscript uses a variant of the Basic language for agent scripting. The known security weaknesses of Basic make it less likely that it will gain traction as a systems management scripting language.

Pinheiro et al. [2] describes a way of using mobile agents for demonstrating aggregation network concepts. They outline a demand driven dataflow for the aggregation and filtering. The aggregation nodes in the dataflow graph are implemented using mobile agents.

Sumatra [13] is extension of Java that supports resource aware mobile programs. The resource monitor component in ASM is similar in concept to the Komodo, a distributed resource monitor.

There are many reported works in mobile agents [16], but we found that there are significantly few reported studies like [15] outlining experimental results and performance measurements. There are even fewer studies that demonstrate mobile agents implementation results over enterprise networks of meaningful size. Secondly, most of the published studies [15] target particular mobile agent systems, whose performance naturally differs depending on implementations. Many of these studies however fail to measure actual benefits of mobility to applications. In our work, we principally focused on creating "agile" systems manager for self-configuration and self-deployment and we measured the extent of benefits that mobility to offers to usual management operations such as discovery and performance management.

9 Conclusion

In this paper, we described why changing enterprise network architectures motivate new agile systems management architecture. We then described the necessary components of agility (e.g. application level auto-configuration, resource and network-awareness), and how an agile systems manager can be constructed over a tuple-space based agent system. Empirical data from a performance management prototype shows an improvement of 40% in the discovery process, and 15% in the performance management task.

Our current work is focused on making ASM pervasive. This involves bring downsizing components of the ASM to fit on resource constrained devices such as cell phones and PDAís. This enables anytime, anywhere management of enterprise networks from thin mobile devices. A particular challenge is supporting significant ìcontrol bandwidthî on mobile devices, whereby they can initiate and monitor complex customized management actions.

References

1. G. Goldszmidt, Y. Yemini. Distributed Management by Delegation. In 15th International Conference on Distributed Computing Systems, IEEE Computer Society, June 1995.
2. R. Pinheiro, A. Poylisher, H. Caldwell. Mobile Agents for Aggregation of Network Management Data, Third International Symposium on Mobile Agents, October 1999.
3. V. Vasudevan, S. Landis. Malleable Services, 34th Hawaii International Conference on Systems Sciences, January 2000.
4. V. Vasudevan, S. Landis, C. Jia, S. Adwankar. Malleable Services. OOPSLA Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems, October 2000
5. K. Thompson. JiniTM Technology Surrogate Architecture Specification. http://developer.jini.org/exchange/ projects/surrogate/
6. W. Keith Edwards. Core JINI. Prentice hall, Sept 1999
7. U. Warrier, L. Besaw, L. LaBarre, B. Handspicker., The Common Management Information Services and Protocols for the Internet. RFC 1095, Internet Engineering Task Force, Oct 1990.
8. S. Adwankar. Mobile CORBA. 3rd International Symposium on Distributed Objects & Applications, September 2001.
9. AdventNet. SNMP Agent Toolkit. http://www.adventnet.com.
10. IBM. TSpaces project home page. http://www.almaden.ibm.com/cs/TSpaces
11. K. McCloghrie, M. Rose. Management Information Base for Network Management of TCP/IP-based internets:MIB-II. RFC 1213, Internet Engineering Task Force, March 1991
12. A. Mohindra, A.Purakayastha, D. Zukowski, M. Devarakonda. Programming Network Components Using NetPebbles: An Earlly Report, Usenix COOTS, April 1998.
13. A. Purakayastha, A. Mohindra. Systems Management with NetScript, LISA, December 1998.
14. A. Acharya, M. Ranganathan, J. Saltz. Sumatra: A Language for Resource-aware Mobile Programs, Spring Verlay Lecture Notes in Computer Science.
15. M. Dikaiakos, M. Kyriakou, and G. Samaras. Performance Evaluation of Mobile-Agent Middleware: A Hierarchical Approach, 5th IEEE International Conference on Mobile Agents, December 2001.
16. A. Bieszczad, B. Pagurek, T. White. Mobile Agents For Network Management, IEEE Communication Surveys, 1998