February 2000

Agents in Network Management

Osman Ertugay
University of Pennsylvania

Michael Hicks
University of Pennsylvania

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

Jessica Kornblum
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-00-09.

Agents in Network Management

Abstract

The ubiquity and complexity of modern networks require automated management and control. With increases in scale, automated solutions based on simple data access models such as SNMP will give way to more distributed and algorithmic techniques. This article outlines present and near-term solutions based on the ideas of active networks and mobile agents, which permit sophisticated programmable control and management of ultra large scale networks.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-00-09.

Agents in Network Management

Osman Ertugay Micheal Hicks Jessica Kornblum
Jonathan Smith

26th February 2000

Abstract

The ubiquity and complexity of modern networks require automated management and control. With increases in scale, automated solutions based on simple data access models such as SNMP will give way to more distributed and algorithmic techniques. This article outlines present and near-term solutions based on the ideas of active networks and mobile agents, which permit sophisticated programmable control and management of ultra large scale networks.

1 Introduction

Although the widely accepted definition of "Network Management" has not changed in over a decade, the importance of managing a network has increased dramatically. Networks have become embedded in almost everything we do from placing a phone call or sending an email to running a business. What was initially a few computers connected together has now become a vast array of specialized devices connected by phone lines, optical cable, satellites, and microwave dishes. The number of network services has also dramatically increased. The telecommunications industry has recently created many "value added services" such as caller identification, call waiting, and call forwarding. They also offer customers a diverse range of network resources such as Plain Old Telephone Service (POTS), Integrated Services Digital Network (ISDN), Digital Subscriber Lines (DSL), T1 and T3 lines. The Internet service base has also expanded, with demand for quality of service (QoS) guarantees, mobile computing, and secure communications. And of course, new services in the network engender the proliferation of new software applications and specialized hardware devices.

As networks and their services and applications become larger in scale and more complex, the need for network management increases. Informally, network management seeks to configure, diagnose, and otherwise control the network and its services. However, traditional models of network management have not evolved at the same rate as the networks they manage. For example, the Simple Network Management Protocol (SNMP)[1]is still the *de facto* standard in the Internet community, even though it was developed over ten years ago when IP-based networks presented a much different landscape. Most network managers find that an ad-hoc collection of scripts is their most valuable management tool, rather than a well-defined approach involving automation, simply because the network is too complex.

Distributed systems and distributed algorithms, have arisen as a way to deal with scale, complexity and diversity in networks. Distributed algorithms are used in current networks in an ad-hoc way; routing algorithms in the Internet are a conspicuous example. While other approaches exist, the two best-known approaches that provide a *platform* for implementing distributed algorithms are agent-based systems and active networks. In both cases, systems across the network are augmented with some level of programmability. In the case of agents, this usually occurs at the application layer, while in active networks the entire network itself becomes programmable, including the network layer using "active packets."

In this paper, we propose to enhance the existing client-server model of network management with an agent-based model that allows distributed computation. In doing so, we aim to reduce the complexity of implementing management tasks, and improve the agility with which these tasks can be deployed to react to new network services and infrastructures. By providing a uniform model for distributed programmability, we believe network management tasks may be implemented with greater

automation. While our thinking has been informed by our experiences with active networking, in this paper we will adopt the mobile agent perspective.

In the following section, we introduce and discuss *agents*, and the advantages of agents as compared with the client/server model. Section 3 examines the current practice of network management and highlights areas that need improving. In section 4, we present a number of approaches to network management using agents, describing work we have already done in the active networking context, as well as work to be done in other agent models. We briefly describe various related efforts in Section 5. We present conclusions in Section 6.

2 Agents

Though many definitions abound, we minimally define an *agent* to be a program that performs a task on behalf of some other entity (like a person or larger program). Expanding on this definition, agents may have a number of characteristics. A *stationary* agent runs locally, perhaps sending messages across the network, while a *mobile* agent may move (both its code and its state) across the network during its computation. An agent may be *persistent*, waiting for events to occur and then responding to them, or it may be *ephemeral*, simply performing its task and then terminating. If agents may communicate with one another, they may be composed into larger applications.

2.1 Benefits of Agents

An agent is used to offload some of the computation that would normally be required of its client. If the client is a human user, then the agent simply automates a common task. For example, in ssh[2], a user may use an ssh-agent to hold his ssh-key's passphrase and automatically supply that phrase whenever the user makes connections through the agent, obviating the need of the user to repeatedly type the passphrase. However, if the client is a program, then the end result is to decompose and distribute the program into autonomous pieces. For example, consider the client-server model of communication as illustrated in Figure 1.

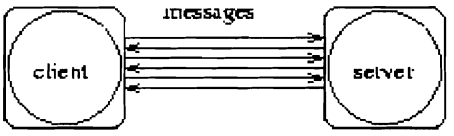


Figure 1: Client-server model

Programs are depicted as circles, and computers are depicted as squares. In this example, a client program wishes to obtain some information located at a server elsewhere on the network. The client then sends a number of request messages to the server, and the server replies with the information. This is a common paradigm in network management: the client is the *NOC (Network Operations Center)*, and the server is a network host or router being managed. The request messages correspond to SNMP *GET* messages and the responses are SNMP *RESP* messages.

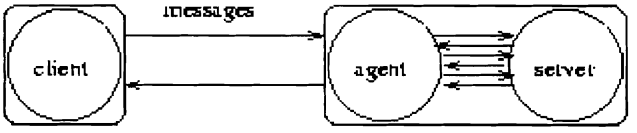


Figure 2: Agent model

The agent model is depicted in Figure 2. Here, an *agent* of the client program is co-located with the server. Upon receipt of the client message, the agent initiates a number of local messages to the server,

and responds to the client based on the server's responses and some computation. This approach has a number of benefits:

- *reduced network traffic*
  Communication between client (via the agent) and server is local until the final result is calculated and returned. This especially important if the management action is to diagnose and respond to an already congested network.
- *distributed computation*
  Some of the client's computation is offloaded to the agent running on the server. This has two benefits. First, this reduces the load and resource consumption on the client application. In the case of network management, we could imagine a single NOC administering hundreds of nodes. By offloading a small computation to each node, we barely disturb the node, but we greatly benefit the client. Second, the resulting distributed computation is more modular and easier to understand.
- *low latency*
  By localizing communication between agent and server, related information is more accurate because of reduced latency. Consider *n* queries made to the server for related information. In the client-server model, each query is separated by a potentially large gap during message transit, such that a (relatively) large amount of time may pass between the first and *(n-1)*th message, diluting the relationship between the information contained in the messages. In contrast, the agent model reduces the latency between each server message, as well as the latency of the overall *macro-*operation.
- *fault tolerance*
  The agent model is more fault tolerant in two ways. First, messages between the server and the agent are local, and thus essentially reliable, in contrast to messages across the network. Second, both the code and data relevant to pieces of a distributed computation are condensed into a single entity: the agent. In the client-server case, there is often data duplicated between client and server, which can become inconsistent during partial failures.
- *secure communication*
  If messages between client and server are to be authenticated and/or encrypted, the benefit of the agent paradigm is magnified, simply because the number of messages that traverse the untrusted network, and thus the number of messages to encrypt/sign, is reduced. Assuming that the agent is authenticated when it is spawned at the server, all of its communications with the server may occur without authentication because they are local.

There are additional benefits if agents are persistent; that is, they reside on the server for an extended period, reacting to events. Two relevant kinds of events are timeouts and server notifications. If agents may wake up at periodic intervals, they may poll the server and assess current conditions, obviating potential network traffic due to client polling. Additionally, they may react directly to server notifications. In the traditional SNMP model, the server may notify the client of events by sending a *TRAP* message. In the agent model, the *TRAP* could instead notify the agent, which may react locally or forward the information to the client. In the former case, we reduce the load on the client and the latency of the response.

Finally, if agents may communicate among themselves, then the client application need be involved even less often. We could imagine an application structured hierarchically, such that certain agents defer to others in coordinating events.

2.2 Challenges of Agents

Agents are not a panacea, however. By increasing the "vocabulary" of the server with agent computation, we increase the potential for malicious or inadvertent damage to the server. In an untrusted setting, we must provide security commensurate with the flexibility available in an agent environment. Fortunately, most researchers in agent technology consider security a solved problem for special-purpose environments[3] by employing cryptographic techniques; more general solutions, such as resource control[4,5] are also possible.

Perhaps more problematic is designing an agent environment amenable to non-experts. In principle, the agent paradigm should allow most anyone to write small bits of computation to act on their behalf, but this is difficult for non-experts in current agent implementations[3]. However, in the area of network management, this issue is less problematic because network managers are not "non-experts." Our experience within the active network context has borne out both of these conclusions, as we describe in more detail in Section 4. In the next section we shall discuss standard network management models to place some context for our proposed solutions.

3 Network Management

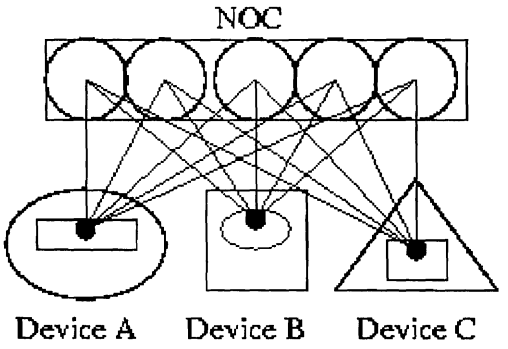
Network Management broadly describes all activities needed to insure a network is behaving as expected. OSI spent a considerable amount of time in the early 1990's defining "Network Management." The following five functional areas have since been accepted as the *de facto* definitions in both the Telecommunication and Internet management communities[6]: *fault, performance, accounting, security,* and *configuration and name management.* Each functional area requires extracting information from managed devices. For example, *fault management* tasks must gather information about the topology of the system such as if an interface or gateway is alive, *performance management* tasks are interested in statistical metrics such as interface load or average queue length and *con*figuration and name management might query device MAC or IP addresses. In the following section, we will describe two popular approaches to managing networks: *ad-hoc* and *centralized.*

3.1 Popular approach

Network management is a constant cycle of monitoring, analyzing and controlling different components of the network. There are two popular approaches to building network management applications. Both approaches use the client/server model.

The first approach, *ad-hoc management*, uses a set of customized, independent scripts. Scripts are usually written to focus on a very small, discrete task such as calculating the load for a given interface or checking configuration parameters for a device. The manager typically runs the script by manually sending it to a managed device. This is a very ad-hoc and popular approach to network management because it is simple. However, ad-hoc management is very costly and inefficient because it places the complexity of managing the network on the manager; manually monitoring and fixing erroneous behavior is very time consuming and inefficient. The positive aspect to ad-hoc scripts is that management tasks are divided up into modular components. Modularity allows the manager to write flexible management applications that can easily be extended as the network evolves.

The second approach to network management is a *centralize* approach. In a centralized approach, one application is developed to handle all five areas of network management. This tactic is



depicted in Figure 3.

Figure 3: Centralized Network Management model

In the depicted network, we have a centralized NOC with three different managed devices. Each device exports some local information. Traditionally, the local information is formatted as a Management Information Base (MIB). It is not necessary that each device supports the same API to access local information and hence the shapes in the figure are different. For example, our application in Figure 3 could query Device A using SNMP, but query Device B and C using *CLI (Command Line Interface).* Our example shows one large application with five management tasks. Each task is responsible for polling the devices for information, making a centralized decision on the network's health and then possibly initiating a control mechanism to change the device's state.

The centralized management model is complex and inextensible. If a new device is added to the managed set or a known device extends its set of exported local information (new MIB or device OS), the NOC must halt management applications and recompile to include new information. It is not an acceptable solution to halt network management software applications for a critical network system. This model also suffers from a limited or unsafe API between the NOC and its managed devices. For example, the SNMP API only allows three types of functions: *get, set* and *trap. Get* and its various forms are used for querying the managed device. *Set* is used as a control mechanism to change state. It is important to note that not all variables are write enabled. *Trap* is used by the SNMP agent (server) to notify the NOC. An SNMP agent is capable of monitoring individual variables for some condition then notifying the NOC once the condition becomes true[1]. Such limited API methods force the NOC to use a polling mechanism to determine the health of the network, which may cause network congestion. A positive aspect to the centralized model is that it naturally provides a global view of the network.

By combining the positive aspects of both the ad-hoc and centralized management models, we can create a management infrastructure that is modular, extensible and capable of managing a network from a global view. Agent technology provides us with the best of both worlds.

4 Agents and Network Management

In this section, we examine more closely how the agent model can be applied to improve automation and address the complexity problem in network management.

A common characteristic for all agent-based models is extensibility. Agents can be deployed and removed dynamically over the network, hence the computation required for a management task can be changed in a flexible manner. Agents nicely accommodate the diversity and dynamism of requirements from network management.

Much of our recent research has examined applying ephemeral, mobile agents to the task of network management. This work has been done in the context of an active, or programmable, network, called PLANet[7]. In PLANet, packets consist not of the header and data fields of traditional packets, but instead, of packet programs, not unlike mobile agents. Packet programs are written in a small, script-like language called PLAN, the Packet Language for Active Networks[4]. PLAN provides simple primitives for data aggregation, control-flow, and computation, but is expression-limited so that all PLAN programs are guaranteed to terminate. In particular, there is no way in the language to express infinite loops or recursive function calls.

In previous work we used PLAN to write small diagnostic programs useful for network management. For example, a simple *ping* program implemented in PLAN is depicted below:

```
fun ping (src:host, dest:host) : unit =
        if (not thisHostIs(dest)) then
          OnRemote(|ping|(src,dest), dest, getRB(), defaultRoute)
        else
          OnRemote(|ack|(), src, getRB(), defaultRoute)
```

The program works as follows. In the case that the packet has not reached its destination, the first case applies, as the call to **thisHostIs(dest)** fails, and so the packet is resent towards its destination using the **OnRemote** primitive. The details of **OnRemote** are unimportant; the key elements to notice are the first and second arguments: **|ping|(src,dest)** indicates that the function **ping**, with arguments

src and **dest**, should be evaluated at the destination **dest**, the second argument. Once the destination is reached, the second case will execute, causing a packet to be sent back to the source, and invoking a small acknowledgement function **ack** (not shown).

PLAN can express more interesting programs as well. For example, we can write *traceroute*, which aggregates the list of nodes traversed between a source and a destination, and even *trace-net*, in which packets fan out and discover the topology of the network, within certain parameters (such as hop distance from the source). Both are useful for discovering inconsistencies in routing tables, and for diagnosing downed nodes and/or links. The interested reader is referred to[8] for a more thorough treatment of PLAN programming.

We have also used PLANet as the basis for more complicated service management functions, such as device queue management (perhaps to implement QoS)[6], and packet filtering[9]. In both cases, we can take advantage of PLANet's *extensibility*, which allows network nodes to be augmented with new, persistent functionality available to PLAN programs, termed *services*. For example, we can load a new queuing service, and allow it to be initiated, configured, and/or terminated by a PLAN management packet. To make sure that arbitrary packets cannot change node parameters, PLAN is augmented with security services[8] that allow authenticated packets to access services commensurate with their level of privilege. For example, a user's packet would not be privileged enough to access the service that allows a node to be extended, whereas a management packet would authenticate itself with the node, and thus gain access to additional services.

We are currently examining how PLAN may be used as a mobile agent technology for management outside of the active network context.

4.1 Management Models

PLANet uses one sort of agent model in implementing network management. Various other models exist based on agent characteristics. These characteristics can be examined along three discrete dimensions: mobility, persistence and cooperation. This classification is a good starting point for analyzing agent-based models in a more structured manner.

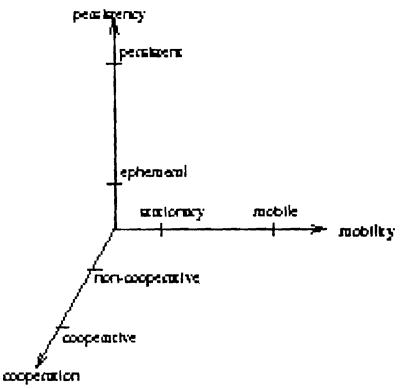


Figure 4: Agent Characteristics

Persistent agents run indefinitely to perform long-term computations. *Ephemeral* agents, on the other hand, perform relatively shorter computations and terminate. Although the distinction may seem vague, one can think of persistent agents as executing in an infinite loop where termination depends on special conditions while ephemeral agents are designed to terminate and loops are bounded. Management tasks that depend on long-term system state are candidates for persistent agents. Ephemeral agents can handle tasks that depend on instantaneous state or no state at all.

Stationary agents perform their computation at a single node. *Mobile* agents, on the other hand, may move from node to node during their computation. Tasks local to a node may be performed by stationary agents while tasks with topological dependencies may be performed by mobile agents. Stationary agents may also perform global tasks if they are persistent and able to cooperate with

remote agents. Hence, the choice between stationary and mobile models depends on the agent characteristics in other dimensions as well as the requirements of the particular management task.

Finally, *cooperative* agents communicate with peer agents at other nodes to perform their computation. *Non-cooperative* agents act alone. Note that in both cases agents may communicate with higher-level management applications. Tasks that require a global view to make local decisions are handled by cooperative agents while tasks that have only local dependencies are handled by non-cooperative agents.

4.2 Applications

There are eight possible models based on the classification in Figure 4. Choice of a model depends on the properties of particular management tasks. The PLANet model described above used the *mobile/ephemeral/non-cooperative* model. In this section, we focus on two other models with potential applications: the *stationary/persistent/non-cooperative* and *stationary/persistent/cooperative* models. We also examine how agents can be used to introduce new network services in the context of the *stationary/persistent/cooperative* model.

Stationary/Persistent/Non-cooperative model

Consider the task of monitoring the running mean and variance of the amount of traffic going out from each interface of each router in our network. This is an informational management task, i.e. agents do not change any operational properties on the routers. The computed information may be used for future system planning, or for (figuring out) load distribution over the network.

This task is best suited by a *stationary/persistent/non-cooperative* model. Agents need to be persistent to periodically get the number of bytes sent out from each interface, update the mean and variance values by using an appropriate algorithm, and log the computed values periodically (much less often than the sampling frequency) to an external application. Mobility and cooperation of agents, furthermore, do not provide any additional benefits. This model has all the benefits described in Section 2 over the traditional management model based on centralized polling.

Stationary/Persistent/Cooperative model

Consider the logging task performed by the agents in the above example. Rather than each agent logging to a central application, in a *cooperative* model, agents can form a hierarchy by using a distributed spanning tree algorithm, and log to their parent agents. In general, the cooperative model allows the agents to choose and form the right communication topology for efficiency in terms of bandwidth, delay, and scalability.

Agents can be used not only as watchers of the network, but as implementers of its services, as proposed by active networks. The *stationary/persistent/cooperative* agent model is well-suited to this task for three reasons. First, because agents are long-lived, they can react to network events, such as timeouts and user- or network-requests. Second, because agents cooperate, large services can be broken down into smaller modular components that are easier to understand. Furthermore, communication may occur with agents at other nodes without loss of abstraction. Finally, services may be easily evolved by deploying new agents or upgrading old ones.

Consider the example of providing guaranteed quality of service (QoS) to network clients. Many routers and switches provide low-level mechanisms (packet filters, priority queues, traffic counters, etc.) for supporting QoS, and higher level protocols, such as *IntServ*[10] and *DiffServ*[11], are built on top of these mechanisms. We believe that agents can be used to implement these services in a modular and extensible manner. Persistent agents at QoS-capable nodes can monitor and control the filtering, classification, queuing and other mechanisms that the underlying device exports. Network wide QoS objectives can be achieved by communicating the QoS state between node agents. Furthermore, the algorithms and policies that govern the allocation of resources can be changed dynamically. Implementing DiffServ with this agent model is especially appealing because the domain can tailor its implementation based on its own constraints and objectives, and can update the system dynamically to accommodate new requirements. We are currently working on the design and implementation of an

agent-based QoS management model on commercial network devices that allow dynamic loading and execution of mobile programs.

5 Related Work

The research community has recognized some of the problems that we have mentioned, and a number of different approaches have been proposed. RMON 1 and 2 (Remote Network Monitoring)[5] were developed to relieve the NOC from load incurred during monitoring. RMON is a specification that presents a higher-level MIB that combines lower-level functions into single entities so as to reduce the load on the NOC, offsetting some monitoring responsibility to the managed device. However, this monitoring ability is fundamentally fixed; the manager could not, for example, specify new filters at runtime.

AgentX[12] was developed with goals similar to ours: to reduce the complexity of management software, and to allow it to change at runtime. These goals are achieved by allowing traditional SNMP agents to be programmed in a modular, extensible manner. AgentX logically splits the traditional SNMP agent in two; a master and one or more subagents. The master agent handles all NOC SNMP requests and passes the information on to the appropriate subagent. Each subagent manages a MIB. If a device or system wants to support new MIB or extend an existing MIB, a new subagent can be loaded and initialized to support the new information. This approach is still fundamentally data-driven, however, and therefore does not offset the complexity of the management application.

Agents have been a popular research topic during the last decade, and a number of agent systems already exist[3]. Several agent systems cite network management as an applicable domain, but none that we know of have developed such applications to an appreciable degree.

6 Conclusion

As networks filter into more aspects of our lives, the importance of the network management system in watching, configuring, and protecting network functions increases dramatically. Current network management models place little burden on the managed device but tremendous burden on the management applications, dramatically increasing their complexity. Furthermore, standard models are fairly rigid, making it difficult to accommodate new services and advances in networking technology. To solve these problems of high complexity and rigidity, and therefore scalability, we have presented new models of network management that move some computational ability to managed devices through the use of *agents*, or more generally, active networks. Our classification of agent models in this context serves as a means for examining the tradeoffs of various approaches. In particular, within this context we have described prior work in the active network system, PLANet[7], as well as presented two other feasible agent models for network management along with example applications.

References

[1] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). Technical Report RFC 1157, IETF Network Working Group, May 1990.

[2] Tatu Ylonen. SSH(1), 1995. Linux 6.1 Man Page.

[3] Dejan Milojcic. Mobile agent applications. In *Trend Wars,* IEEE Concurrency, pages 80-90. IEEE, July-September 1999.

[4] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pages 86-93. ACM, September 1998.

[5] Paul Menage. RCANE: A Resource Controlled Framework for Active Network Services. In *Proceeding of the First International Working Conference on Active Networks (IWAN'99), July 1999.*

[6] William Stallings. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. Addison-Wesley, third edition, 1999.

[7] Michael Hicks, Jonathan T. Moore, D. Scott Alexander, Carl A. Gunter, and Scott Nettles. PLANet: An active internetwork. In *Proceedings of the Eighteenth IEEE Computer and Communication Society INFOCOM Conference,* pages 1124-1133. IEEE March 1999.

[8] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. Network programming with PLAN. In Luca Cardelli, editor, *Proceedings of the IEEE Workshop on Internet Programming Languages,* volume 1686 of *Lecture Notes in Computer Science*, pages 126-143. Springer-Verlag, May 1998.

[9] Michael Hicks and Angelos D. Keromytis. A secure PLAN. In Stefan Covaci, editor, *Proceedings of the First International Workshop on Active Networks*, volume 1653 of *Lecture Notes in Computer Science,* pages 307-314. Springer-Verlag, June 1999. Extended version at http://www.cis.upenn.edu/~switchware/papers/secureplan.ps.

[10] IETF Integrated services working group, 1999. http://www.ietf.org/html.charters/intserv-charter.html

[11] IETF Differentiated services working group, 1999. http://www.ieft.org/html.charters/diffserv-charter.html

[12] M. Daniele and B. Wijnen. Agent extensibility (AgentX) protocol. Technical Report RFC 2741, IETF Network Working Group, January 2000.