An Active Network Approach to Efficient Network Management

Danny Raz Yuval Shavitt

Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733-3030
raz@research.bell-labs.com, shavitt@ieee.org.

Abstract. Active networks is a framework where network elements, primarily routers and switches, are programmable. Programs that are injected into the network are executed by the network elements to achieve higher flexibility and to present new capabilities.
This work describes a novel active network architecture which primarily addresses the management challenges of modern complex networks. Its main component is an active engine that is attached to any IP router to form an active node. The active engine we designed and implemented executes programs that arrive from the network and monitors and controls the router actions. The design is based on standards (Java, SNMP, ANEP over UDP), and can be easily deployed in todays IP networks.
The contribution of this paper is the introduction of novel architectural features such as: isolation of the active mechanism, the session concept, the ability of active sessions to control non-active packets, and blind addressing. Implementing these ideas, we built a system that enables the safe execution and rapid deployment of new distributed management applications in the network layer. This system can be gradually integrated in todays IP network, and allows smooth migration from IP to active networking.

1 Introduction

The emerging next generation of routers exhibit both high performance and rich functionality, such as support for virtual private networks and QoS [11]. To achieve this, per flow queueing and fast IP filtering are incorporated into the router's hardware [11]. The management of a network comprised of such devices and efficient use of the new functionality introduces new challenges.

Active networks is a framework where network elements, primarily routers and switches, are programmable [14]. Programs that are injected into the network are executed by the network elements to achieve higher flexibility for networking functions, such as routing, and to present new capabilities for higher layer functions by allowing data fusion in the network layer.

This work suggests a novel active network architecture which primarily addresses the network management challenges. At its center, an active engine executes programs that are received through the network. The active engine is attached to an IP router, and together they form an active node. We introduce the notion of a *session*, which generalizes the soft state mechanism that appears in many active network architectures [10, 12]. This enables long lasting applications that are typical to network management, to reside in the active node. For a scalable distributed operation we introduce a new addressing mode, *blind* addressing, in addition to the *explicit* mode available in IP. We explicitly allow sessions to access the router management information base (MIB) using SNMP. Overall, we introduce a modular solution that enables an easy deployment in the current IP networks.

This paper describes a working network prototype. The active engine is written mostly in C, and is demonstrated on a network comprised of software routers running on FreeBSD PCs and a Cisco 2514 router with a PC as an adjunct active engine. We put an emphasis on standard APIs and tools. The mobile code is written in Java, it is encapsulated together with data using the standard ANEP headers [2] over UDP. The engine communicates with the router using SNMP which enables it to monitor and control the router's operation.

Our approach can handle the entire range of active networking, from capsules to programmable switches. Capsule applications carry their code and terminate after execution. Programmable switches are implemented by "well-known" session ids that may receive data and act on it. We also allow authorized sessions to intercept non-active packets and manipulate their data, change their routing, drop them, etc. Such authorized sessions can also change the MIB variables in the router using SNMP. Another feature of the architecture is the ability of non-active packets to request a special service, e.g., routing, that is implemented by a transparent resident session that is mapped to this service at the router.

Network management applications are traditionally centralized around some manager. The manager queries the managed objects, builds a view of the network, and sends alerts if a problem is detected. The manager can also try and take corrective actions by sending configuration commands to network entities. The recent trend in network management architectures is to rely on multiple levels of abstraction, e.g., CORBA, Java ORB, Java RMI, Styx, DCOM, and Directory Enabled Networks (DENs). As a result, the cost of management is obscured from the application programmer, and thus neglected. If this trend will continue, management may consume increasing portions of network resources (bandwidth, buffer space). As is well put in the last chapter of [15]: *"When CORBA is used the wrong way, the implemented applications, although they are functionally complete, can have performance and scalability problems."* We believe that our approach presents a better alternative to the current practice. It calls for the distribution of the management task in the network, enables shorter control loops, deletes long haul dissemination of redundant and unimportant information ("I'm OK" messages), and facilitates new exciting applications. The framework forces the programmer to be aware of efficiency issues and thus will result in more efficient code not only due to its intrinsic capabilities to do so, but also due to the human change of focus. Other agent based approaches [8, 9] that enable distributed computing rely heavily on bandwidth blind approaches such as Java RMI and thus do not result in efficient usage of network resources.

The rest of the paper is organized as follows. Section 2 gives a short overview of the system we built. A detailed comparison between our design and existing systems appears in Section 3. Section 4 describes the system architecture and flow of information. Section 5 shortly describes implementation examples. We discuss future work and give our concluding remarks in section 6.

2 System overview

Logically, an active node in our system is comprised of two entities: an IP router, and an adjunct active engine (AE). The IP router component performs the IP forwarding, basic routing, and filtering that are part of the functions performed by today's commercially off-the-shelf (COTS) IP routers. The filtering is used to divert packets to the active engine. The active engine is an environment, where user written programs can be executed with close interaction to the router data and control variables[1].

Physically, the IP router and the active engine may either reside on different machines or coreside inside the same box. This structure enables us to upgrade any COTS IP router to an active router simply by adding an adjunct active engine. The separation protects non-active traffic from the effects of erroneous operations of the active part of the network, and inflicts minimal additional delay on non-active traffic. It also makes gradual deployment of active nodes in current networks easy.

A logical distributed task is identified by a globally unique number called a session id. When code associated with a non-existing session arrives, it is executed and creates a process that handles all the packets of that session. Such a process can either handle only a single data packet and terminate (capsule), or it can exist in the AE for a long period of time handling many data packets as required by many network management applications.

To perform network layer tasks, sessions must have access to the router's network layer data, such as, topological data (neighbor ids), routing data, performance data (packets dropped, packets forwarded, CPU usage etc.) and more. We use SNMP as the interface between the router and the AE. Standard SNMP agents exist in all routers and enable a read/write interface to a standard management information base (MIB).

In order to perform distributed tasks, an active node must have means to communicate with other active nodes. Relying on the fact that the full topology information is available for any specific node does not scale. To tackle this problem we support a topology-blind addressing mode that enables a node to send a packet to the nearest active node in a certain direction. This mode is useful for topology learning, robust operation, support of heterogeneous (active

[1] The AE can be perceived as an execution environment in the context of [6].

and non-active) environment, etc. We also support the explicit addressing mode in which a packet is sent to a specific active node.

Overall, we built a system that enables the safe execution and rapid deployment of new distributed management applications in the network layer. This system can be gradually integrated in todays IP network, and allows smooth migration from IP to active networking. To facilitate this, we introduce novel architectural features such as: isolation of the active mechanism, the session concept, the ability of active sessions to control non-active packets, and blind addressing.

3 Related work

Recently, research in active networking is gaining popularity. Some of the research groups in this area are: MIT (ANTS) [16]; U. Kensas [10]; U. Penn. (SwitchWare) [1]; Georgia Tech. [4]; Columbia (NetScript) [17]; UC Berkeley (MeGa) [3]; Washington University (DAN) [5], and more. In this section we compare our architecture with the menagerie of existing active network architectures and with other agent based approaches.

In our design, we separate the active engine where active code is executed from the router itself. This approach makes deployment easier and poses less threat to the non-active traffic in case the active engine breaks down. A similar approach was recently reported by Amir et al. [3], however, they are limiting the scope of their active server to the application level and thus limiting its capabilities. Bhattacharjee et al. [4] also suggested a similar approach but for a very restricted active server that can support only a given set of functions. In most other works, the active part and the non-active part are not well separated.

ANEP [2] is incorporated into several of the existing projects. It is already used in [10] and will be used in [16]. We can only speculate whether the current research projects will converge eventually to a unified environment using ANEP as the wire encapsulation. Interestingly, SNMP escaped the notice of all the active network projects but one [17]. We believe the use of SNMP is the most attractive option to integrate active network technology with existing routers.

Most agent based systems [8, 9] reside in the application layer and thus do not have access to network layer information. A recent first step in addressing the need of agents to interface with network layer information is presented by Zapf et al. [18]. They allow their application layer agents to access router information through an intermediate resident application in the router using SNMP interface. Another interesting work was presented by Hjálmtýsson and Jain [7]. They built a system where installed agents can manipulate data streams in a router. Though similar in flavor to this work, Hjálmtýsson's work suggests a new router design, while we emphasize the use of legacy routers.

4 Architecture

4.1 Design principles

Targeting specifically the network management domain, the following principles guided our design:

Generality and Simplicity Building applications should be easy to a large base of programmers. Thus, the system should not be limited to one language, and should support languages that are in general use. The node should also be general enough to support many levels of active networking, from capsules to programmable switches.

Modularity We separate the active node to modules with clearly defined API between them. In particular, we chose to separate the forwarding mechanism of a regular router from the operating environment where the packets are executed. We also use, as much as possible, well accepted standards, such as Java, SNMP, and ANEP [2], as the API in which the modules exchange information.

Inter-operability and heterogeneity Most likely, active nodes will co-exist with non-active routers. Furthermore, incremental deployment of active nodes with co-existing routers seems a natural evolvement path. In such a scenario it is very unlikely to assume that an application running on an active node could explicitly know the addresses of its active neighbors. To this end we support "blind" addressing, in which the active node needs not know the address or the location of other active nodes.

Long Lasting Sessions In many network management applications there is a natural need for an application to reside in a node for a long period of time (for example to do monitoring and billing). This cannot be efficiently implemented with capsules. As we do target the network management domain we specificly design the system to support such applications. It is also very important for an application to have an easy and standard access to the local information at a node, since in many applications the action taken by the packet depends on this information.

Cost Visibility Although we wish to abstract most of the technical details in order to simplify the development of applications, we think that the application must be aware of the costs, both in terms of node resources (CPU, memory, etc.), and in terms of global network resources (bandwidth and delay). Therefore, we do not use advanced distributed tools such as CORBA and Java RMI, which in general hide much of the actual cost from the user.

Safety and Security Non-active traffic should not be affected by the new active ability. Further more, an active application should not be able to affect any other application. The system should support security and robustness at all levels.

The above principles directed us to make the following architectural decisions: (1) The active node is composed of a regular router with a *diverter*, which detects and diverts active packets to the main separate component - the Active Engine;

and (2) The AE is a separate entity (which may reside on a different card, or a separate machine), which performs most of the active node's task.

This simple modular structure supports inter-operability, and does not require that the specific address of the next active hop will be known. The diverter part is fairly simple, and can be carried out using IP filtering which is supported in the API level by most of the router vendors. This structure also allows an easy incremental deployment in heterogeneous networks. Another advantage of this design is robustness; non-active traffic could not be effected by active traffic. Even if for some reason the active engine stops working the router will still route non-active packets correctly.

The second significant entity in our design is the session which serves both as a mechanism that preserves soft state, and as a rendezvous point for data fusion. Logically, a session is a distributed task preformed in the network. A session has a unique network id, thus, different programs, on various nodes can belong to the same session. These programs may exchange information using active data packets, and they can distribute (and/or update) their code by sending active programs. This notion of a session is general enough to support both long lasting processes, and short term capsules. The fine details of the design are described in the following subsections.

4.2 Detailed design

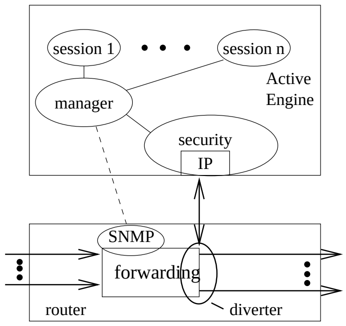

Fig. 1. The general architecture.

The main components of the system are (see figure 1):

Diverter — A part of the router that enables it to divert packets to the AE based on their IP/UDP header. The new generation of high-performance IP routers [11] has this option implemented as part of the router hardware. Edge routers and our prototype perform this function in software.

Active Manager — The core of the AE is the Active Manager. This part generates the sessions, coordinates the data transfer to and from the sessions, and cleans up after a session when it terminates. While a session is alive, the Active Manager monitors the session resource usage, and can decide to terminate its operation if it consumes too much resources (CPU time or bandwidth) or if it tries to violate its action permissions.

Security Stream Module — This module resides in kernel space below the IP output routine. Every connection that the session wishes to open, must be registered with this module to allow monitoring of network usage by sessions. The registration will be done by our supplied objects transparently to the application developer. The module is not fully implemented, yet.

Router Interface — This module allows sessions to access the router Managed Information Base (MIB). It is implemented as a Java object that communicates with the router using SNMP. In the future we plan to enhance performance by caching popular MIB variables.

The design allows multiple languages to be implemented simultaneously, but since the current implementation handles only Java packets we will restrict the description to the details of the Java implementation. Implementation of other languages may require some adaptations according to the language specifics.

In the following we describe the flow of packets through the system. Note that a non active packet does not pass through the Active Engine since the diverter recognizes it as such and thus the packet takes the fast-track to its output port.

All active packets include a default option that contains among others the unique session id of the packet, and a content description (data, language). All the diverted packets are sent to the active manager. If a packet does not belong to an existing session and it contains code it triggers a creation of a session. If it is a data packet it is discarded. A session creation involves, among others, authentication (not implemented), creation of a control block for the session, creation of a protected directory to store session files, opening of a private communication channel through which the session receives and sends active packets, and execution of the code.

Methods that are associated with the session object allows the Java program to easily send itself to another node, and to send and receive data. New arriving programs are passed to the session to allow it to perform code updating without losing its state.

Four UDP port numbers (3322-5) are assigned to active network research. The first, the *blind* addressing port, is used to send active packets to an unspecified node in a certain direction, i.e., towards some distant destination. The diverter in the first active node that is on the route to that destination intercepts the packet and sends it to the active engine. Therefor, the sender is not required to know the address of the next active node. The second UDP port number (the *explicit* active port) is used to send an active packet to a specific active node. This packet is forwarded through the fast-track of all the intermediate active nodes, and is not diverted until reaching its destination.

The active manager keeps track of the resource consumption of the session in the node (CPU time, bandwidth, disk space). A session that consumes excessive resources is aborted. A session may also be aborted due to lack of activity.

Since we expect most network programming to be stable, we do not try to optimize the capsule model. Thus, we are less concerned about the program size as programs are not going to be transmitted frequently. A mechanism to reassemble a program from a chain of UDP packets is currently implemented.

4.3 Security

Security and safety are of major concern in the deployment of active networks. A system is safe if no application can destroy or damage the appropriate execution of other applications. In particular, the active engine as a whole should not effect the routing of non-active packets. A system is secure if all operations including access to data are authenticated, i.e., only authorized sessions can perform actions and or access private data.

Our architecture supports both security and safety, although currently it is not fully implemented. In any design, one faces the dilemma of choosing between the freedom to allow more sophisticated session behavior (e.g., setting MIB variables, diverting non-active packets) and the fear of a possible safety/security hole. Our approach allows multiple levels of security via authentication and session classification. Each session is authorized to use specific services (MIB access for read or write, divert non-active packets) and resources (CPU time, bandwidth, memory). As it is important to ensure both safety and security in order to promote the use of active network, one can initially select to be more restrictive in authorizing services, and gradually allow more sophisticated services.

Our first concern is to make sure that non-active packets are not affected by the active packets. This is easily achieved by the logical (and sometimes physical) separation of the active engine from the router. Next step in safety, is to ensure that a session will not corrupt or even gain access to other sessions data. We achieve this through the use of Java SecurityManager. It allows us to control the session running environment, in particular we prevent sessions from using native methods and restrict the use of the file system.

Malicious or erroneous over use of system resources is of great concern. To this end, we intend to monitor the use of CPU time by sessions. We implemented a tight control over the usage of the communication channel to the outside world. TCP connections can be only opened by a permitted session using our supplied methods that monitor the bandwidth consumption. An attempt to use Java methods is blocked by controlling the IP layer in the active engine. An unauthorized connection will be dropped. UDP packets can be sent only through the manager, which again can monitor the bandwidth usage.

4.4 Test bed description

We built a small heterogeneous network as described in figure 2. The network is comprised of both FreeBSD based active routers and COTS routers (currently we use CISCO 2500 routers and Lucent Technologies RABU PortMaster3) with adjunct active component. The FreeBSD routers are PCs running FreeBSD, using `routed` for routing. In these PCs, the active engine and the router co-reside in the same machine.

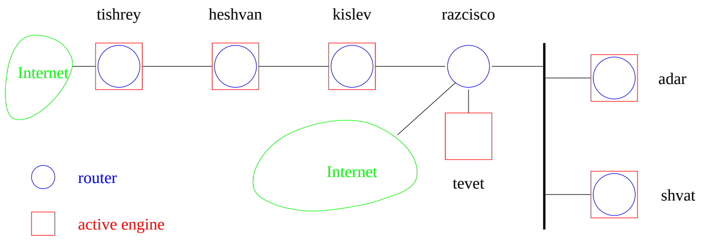

Fig. 2. The prototype network architecture.

Performance In building the prototype we did not aim at performance. Our first target was to build a concept system that will enable us to test design ideas and applications. Thus, many parts of the system where not optimized for performance. Nevertheless, we tested the capabilities of our prototype. Thus the delay of an active packet through the system should be treated as an upper bound on what can be accomplished and the load measures as lower bound.

Our first experiment was to measure the delay of an active packet through one active node. To this end we used a session on heshvan that forwards every packet it receives to the destination on the packet. Java applications on `tishrey` and `kislev` (see figure 2) exchange UDP packets using either an active port that is diverted to `heshvan`'s active engine, or a non-active port that is forwarded by `heshvan`'s router. The packets were about 500 bytes long, and the network was kept without additional traffic to prevent queueing delay to effect the results.

The average round trip delay (RTD) for a packet was 1.37ms without diversion, and 11.20ms with diversion. The 90% confidence interval are 1.37±0.0047ms (∼0.68% wide), and 11.20±0.022ms (∼0.39% wide). Thus the average delay through `heshvan`'s active engine was $(11.2 - 1.37)/2 = 4.915$ms. Heshvan is a Pentium machine with 64Mbytes memory, that runs at 200MHz.

It is obvious that the active engine's performance is bounded by the memory access speed and not by the computation power of the processor. We did not conduct a full scale stress test, but we repeated the above experiment with ten sessions active in `heshvan`, and the delay through the active engine did not change.

5 Application examples

To demonstrate the power of our system, we consider two problems both related to network management but represent basic problems that can be used in various applications. The first one is bottleneck detection, which is a special case of collecting information or calculating a function along a route between two nodes. The second application is a message dissemination for a large group of receivers. It is useful for an automatic configuration of network elements or any other application that require dissemination of messages to a large population. In this section, we briefly discuss the bottleneck detection applications and refer the reader for more details to [13].

Bottleneck detection is an important problem faced in network management. It is a building block for higher level applications, e.g., video conferencing, that require QoS routing. It is also an example for any problem related to gathering information along a given path between two network nodes.

In today's IP networks there is only one ad-hoc technique to examine one specific QoS parameter, namely the delay along a path. It is the well-known `traceroute` program that enables a user at a host to get a list of all the routers on the route to another host with the elapsing time to reach them. The use of the `traceroute` program for network management has several drawbacks: it can only retrieve the hostname and the delay along a path; it is extremely inefficient in its use of network resources; and it is slow.

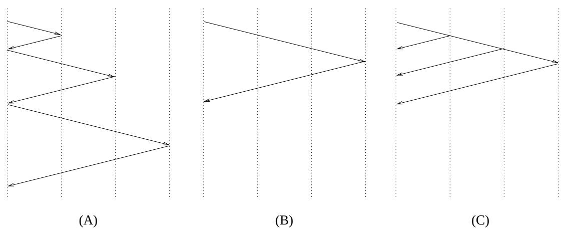

Fig. 3. Three traceroute executions on a three-hop path: (A) the current program; (B) collect-en-route; and (C) report-en-route

In an active network, and specifically in our architecture, there are several options to gather information along a given path between two network nodes, each optimize a different objective. One option (collect-en-route) is (see figure 3(B)) to send a single packet that traverses the route and collects the desired information from each active node. When the packet arrives at the destination node, it sends the data back to the source (or to any management station). This design minimizes the communication cost since a single packet is traveling along each link in each direction.

Another option (report-en-route) is (see figure 3(C)) to send a single packet along the path. When the packet arrives at a node, it sends the required information back to the source and forwards itself to the next hop. This design minimizes the time of arrival of each part of the route information, while it compromises communication cost.

Note that the `traceroute` program has (see figure 3(A)) time and communication complexities that are quadratic in the path length. Table 1 compares the three options. The use of general programs in the capsule enables the application

Algorithm Used	No. of messages used	time of data arrival from node i
`traceroute`	$n(n+1)$	$i(i+1)$
collect-en-route	$2n$	$2n$
report-en-route	$n(n+3)/2$	$2i$

Table 1. Performance comparison (time is measured in hop count).

programmer to query any available (MIB) variable from the router, rather than just the router IP address. For example, for bottleneck detection we can collect statistics about TCP packet loss along a route to a certain host. It is easy to generalize the program to start the data collection from any active node in the network (not necessarily the originator), and to send the reports to any other node.

6 Discussion and future work

Since the inception of the active network idea there is a search for the "killer application", the one that will strongly require active network technology. We believe that network management is a domain where using active network technology could be proved to be very significant. Applications like adaptive control, router configuration, element detection, network mapping, security management (intruder detection, fighting deny of service attacks) are only some examples where active network technology can be successfully applied. Our architecture also supports solutions to other problems not necessarily part of network management, such as, search worms, smart mail, multicast, hop to hop flow control, etc.

The additional delay seen by packets in active networks, is an issue that was addressed in the past, and as others pointed out we believe the small slow-down is compensated by the big saving that can be achieved in traffic volume. The analysis presented in section 5 is a first step towards a more formal framework to address this point. In our prototype, non-active packets suffer only the negligible additional delay of the diverter. And although the active engine is not optimized in the current version, as the emphasis is on functionality, the delay through the AE is reasonable.

Altogether, we presented a prototype implementation of a network management engine built using active network technology. We believe our approach can enable new and efficient ways to manage today's (and tomorrow's) networks.

References

1. D. S. Alexander et al. The SwitchWare active network architecture. *IEEE Network*, 12(3):29–36, May/June 1998.
2. D. S. Alexander et al. The active network encapsulation protocol (ANEP). URL `http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt`, 1997.
3. E. Amir, S. McCanne, and R. Katz. An active service framework and its application to real-time multimedia transcoding. In *SIGCOMM'98*, Sept. 1998.
4. S. Bhattacharjee, K. Calvert, and E. W. Zegura. An architecture for active networking. In *HPN'97*, Apr. 1997.
5. D. Decasper and B. Plattner. DAN: Distributed code caching for active network. In *INFOCOM'98*, Mar. 1998.
6. AN Working Group. Architectural framework for active networks. URL `http://www.cc.gatech.edu/projects/canes/arch/arch-0-9.ps`, August 31 1998. Version 0.9.
7. G. Hjálmtýsson and A. Jain. Agent-based approach to service management - towards service independent network architecture. In *IFIP/IEEE IM'97*, pages 715 – 729, May 1997. San Diego, CA, USA.
8. G. Karjoth, D. B. Lange, and M. Oshima. A security model for aglets. *IEEE Internet Computing*, 1(4):68–77, July/August 1997.
9. J. Kiniry and D. Zimmerman. A hands-on look at java mobile agents. *IEEE Internet Computing*, 1(4):21–30, July/August 1997.
10. A. B. Kulkarni, G. J. Minden, R. Hill, Y. Wijata, S. Sheth, H. Pindi, F. Wahhab, A. Gopinath, and A. Nagarajan. Implementation of a prototype active network. In *OPENARCH'98*, pages 130–143, Apr. 1998.
11. V. P. Kumar, T. V. Lakshman, and D. Stiliadis. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, 36(5):152–164, May 1998.
12. E. L. Nygren. The design and implementation of a high-performance active network node. Master's thesis, Massachusetts Institute of Technology, Feb. 1998.
13. D. Raz and Y. Shavitt. An active network approach to efficient network management. Technical Report 99-25, DIMACS, May 1999.
14. D. L. Tennenhouse et al. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan. 1997.
15. A. Vogel and K. Duddy. *JAVA Programming with CORBA*. Wiley, 2nd ed., 1998.
16. D. Wetherall et al. ANTS: A toolkit for building and dynamically deploying network protocols. In *OPENARCH'98*, pages 117–129, Apr. 1998.
17. Y. Yemini and S. da Silva. Towards programmable networks. In *IFIP/IEEE Intl. Workshop on Distributed Systems Operations and Management*, Oct. 1996.
18. M. Zapf, K. Herrmann, K. Geihs, and J. Wolfgang. Decentralised SNMP management with mobile agents. In *IFIP/IEEE IM'99*, May 1999.