Active Networks for Efficient Distributed Network Management

Danny Raz and Yuval Shavitt, Bell Laboratories, Lucent Technologies

ABSTRACT

The emerging next generation of routers exhibit both high performance and rich functionality, such as support for virtual private networks and QoS. To achieve this, per-flow queuing and fast IP filtering are incorporated into the router hardware. The management of a network comprising such devices and efficient use of the new functionality introduce new challenges. A truly distributed network management system is an attractive candidate to address these challenges. In this article we describe how active network techniques can be used to allow fast and easy deployment of distributed network management applications in IP networks. We describe a prototype system where legacy routers are enhanced with an adjunct active engine, which enables the safe execution and rapid deployment of new distributed management applications in the network layer. This system can gradually be integrated in today's IP network, and allows smooth migration from IP to programmable networks. This is done with an emphasis on efficient use of network resources, which is somewhat obscure by many of today's high-level solutions.

INTRODUCTION

The overwhelming majority of IP network management systems are centralized around some management station. 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 to take corrective actions by sending configuration commands to network entities.

There are many drawbacks to a centralized architecture, which become more evident with the growth in network size and complexity. As the number of controlled elements grows, the requirements for computational power from the management system and bandwidth from the network that connects it grow, too. In a large network, some of the controlled entities are distant from the management station; thus, control loops have long delays, and control traffic wastes bigger portions of the network bandwidth.

To alleviate the scalability problems, distributed control architectures have been proposed in recent years. Most of these solutions delegate some of the central management tasks to distributed software agents or remote objects. The use of distributed object paradigms abstracts the implementation details. Abstractions, such as Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Remote Method Invocation (RMI,) are helpful in designing and building distributed systems, but they hide the true cost of the implementation details. As a result, such distributed systems tend, in many cases, to be inefficient in their use of network resources, primarily in their use of bandwidth. Another drawback of agent-based systems is that this type of delegation does not support a truly distributed agent system where agents can communicate with their neighbors to efficiently carry out distributed tasks. Our approach to network management calls for *efficient* distribution of the management task in the network.

Another important issue in distributed agent systems for management and control is the location at which these agents reside. Clearly, this has a major impact on the performance since it affects the delay in the control loop. In general, software agent solutions assume the existence of available hosts to run their application-level programs. An optimal location for an agent would be in the router kernel, where all the necessary local information is available, and action can be taken locally. However, such a solution is deemed impractical due to the inability to interfere with the router real-time operation constraints. Clearly, the closer the agent is to the controlled system, the better it can perform.

In this work, we propose an architecture that allows the easy deployment of distributed network management applications in IP networks. The architecture encourages the efficient use of network resources, but still maintains the ease of application development. We use the active network as a framework which allows the distribution and execution of network management applications in the network routers.

In our architecture a node comprises two logical entities: a forwarding IP mechanism and a general execution environment that we call the *active engine* (AE). The forwarding IP mechanism is responsible for the task currently done by IP routers, namely, forwarding an IP packet according to local tables. The active engine is a user-level execution environment adjunct to the

0163-6804/00/$10.00 © 2000 IEEE

forwarding mechanism. A well-defined interface allows the active engine to both monitor and control the forwarding component.

This architecture allows us to simultaneously run multiple distributed network management applications. A *session* is a collection of programs (agents) that are injected to the network by authorized users and executed in the active engines. The agents can migrate from node to node and duplicate themselves. Agents belonging to the same session in different nodes can cooperate in their work by exchanging data messages. Using a well-defined network layer interface, namely a management information base (MIB), authorized agents can gather local information and control the forwarding operation. Other software distribution mechanisms can be integrated as well; in particular, popular management programs can be grouped into libraries that will be part of the active engine.

A major concern in adding a distributed control mechanism for IP is security and safety. A system is safe if no application can destroy or damage the appropriate execution of other applications. 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). In our architecture, safety is achieved by separating the execution environment from the forwarding mechanism, and using a well-defined application programming interface (API) between them. Our architecture uses several security and safety mechanisms, discussed later.

An important issue in deploying a scalable distributed network application is the addressing scheme to be used. Agents should be able, of course, to send messages to a known destination. However, in many cases, agents do not know the addresses of remote agents with which they wish to communicate. This may be due to the size of the network, or changes in network topology. Naming systems such as those used by CORBA can help communicating in such scenarios, but the cost of using them is high and often unpredictable. In contrast, we use a very simple and efficient addressing mode that allows agents to send messages to the nearest relevant agents. It is based on the ability of the forwarding mechanism to intercept specially marked packets and redirect them to the active engine.

The logical separation of the router allows us to define a uniform environment and API for management programs (agents). Our active engine isolates the software agents from the vendor-dependent router API. It can also help in the analysis of algorithm performance. Physically, the forwarding mechanism and active engine may either reside on different machines or coreside inside the same box. This structure enables the upgrade of any commercial off-the-shelf IP (COTS) router to a node in our architecture simply by adding an adjunct active engine. This presents a natural evolution path from today's IP networks to future manageable networks.

We implemented our architecture and built a working network prototype. The active engine is written mostly in C, and is demonstrated on a network comprising software routers running on FreeBSD PCs and COTS routers (currently CISCO 2500) with an adjunct active engine. Currently we support Java as the programming language for the active code, but the architecture is built with handles to allow the use of other languages.

The rest of the article is organized as follows. We give a short overview of the system we built. We discuss related work in the fields of network management and active networks. We describe the system architecture and flow of information. We demonstrate the architecture capabilities with two implementation examples. We then discuss future work and give our concluding remarks.

A System Overview

In this section, we provide a general overview of our architecture. For convenience we use active network terminology. In particular, our network node is called an *active node*, and the packets that carry the agents' code and the communication among them are called *active packets* (this traffic is termed *active traffic*).

As mentioned before, an active node in our system comprises 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 COTS IP routers. The IP filtering enables the diversion of active packets (or other packets specified by an authorized session) to the active engine.

The active engine is an environment in which code encapsulated in active packets can be executed. This code can specify how code and data related to a specific task should be handled. A logical distributed task is identified by a globally unique number called a session id (the unique id is a combination of the session originator IP number and a locally assigned unique number). When code associated with a nonexisting 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 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 Simple Network Management Protocol (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 MIB.

In order to perform distributed tasks, an active node must have some means to communicate with other active nodes. Maintaining full topology information at all the nodes 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 and nonactive) environments, and so on. We also support the explicit addressing mode in which a packet is sent to a specific active node.

The active engine is an environment in which code encapsulated in active packets can be executed. This code can specify how code and data related to a specific task should be handled.

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.

RELATED WORK

The first generation of distributed network management systems used delegation to alleviate the load of the management station by allowing the manager to run processes in remote locations [1]. This paradigm has a strict parent-child (or client-server) relationship between the manager and its agents. The next generation used CORBA to handle distributed tasks [2]. Later, distributed object infrastructures such as DCOM and Java RMI use the same basic ideas for object communication.

Most agent-based systems (not necessarily for network management) [3] 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 was presented by Zapf *et al.* [4]. They allowed their application layer agents to access router information through an intermediate resident application in the router using SNMP. Other interesting work was presented by Hjálmtýsson *et al.* [5]. They built a system where installed agents can manipulate data streams in a router, and suggested a new router design. Although similar in flavor to this work, Hjálmtýsson's work suggested a new router design which is not easily deployable in current networks.

Most of the work on active networks [6] aimed to replace the current IP paradigm and thus neglected IP-based applications. While the desirability of this is debatable, our aim is to use active network techniques to manage IP networks and gradually enhance them with mature active network techniques. Similar goals were sought in the BBN Smart Packet project [7], which uses small one-packet scripts for management applications. This project is restricted by the small maximum program size and the lack of soft state maintenance.

In our design, we separate the active engine from the router forwarding mechanism. A similar approach was recently reported by Amir *et al.* [8]; however, they limit the scope of their active server to the application level, and thus limit its capabilities. Bhattacharjee *et al.* [9] also suggested a similar approach, but for a very restricted active server that can support only a given set of functions. In most other work, the active and nonactive parts are not well separated.

Interestingly, SNMP escaped the notice of most active network projects except a few (e.g., [10]). In general, most of the research on active networks [6] failed to notice the need to access local network layer information. We believe the use of SNMP is the most attractive option to integrate active network technology with existing routers.

ARCHITECTURE

DESIGN PRINCIPLES

Our goal is to design a system that allows easy and fast deployment of distributed network management applications. The following principles guided our design.

Generality and Simplicity — Building applications should be easy for a large base of programmers. Thus, the system should not be limited to one language, and should support languages in general use. The node should also be general enough to support both long-- and short-lived applications.

Modularity — We divide the node into modules with a 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 Active Network Encapsulation Protocol (ANEP) [11], for the API.

Interoperability and Heterogeneity — Most likely, active nodes will coexist with nonactive routers. Furthermore, incremental deployment of active nodes with coexisting routers seems a natural evolution path. In such a scenario it is very unlikely 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 need not know the address or location of other active nodes.

Network Layer Interface — It is very important for an application to have 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. This access should support read and write operations since a management application should be able to take corrective actions.

Cost Visibility — Although we wish to abstract most of the technical details in order to simplify the development of applications, we believe that the application must be aware of the costs, in terms of both node resources (CPU, memory, etc.) and 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 programmer.

Safety and Security — Adding new functionality should not affect the legacy network operation. Furthermore, an unauthorized management application should not be able to affect any other application. The system should support security and robustness at all levels.

In view of the above principles, we chose to use the active network technology for our needs. This technology allows a flexible execution environment for management applications, without the need to define new protocols and standards. This resulted in the architecture described earlier. This simple modular structure supports inter-operability, and does not require the specific address of the next active hop to be known. This structure allows easy incremental deployment in heterogeneous networks, and is also robust since no-active traffic cannot be affected by active traffic. A significant entity in our design is the session. Logically, a session is a distributed task performed in the network. A session has a unique network id; thus, different programs on various nodes can belong to the same session, and exchange

information using active data packets. 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 below.

DETAILED DESIGN

The main components of the system are (Fig. 1):

- Diverter — A part of the router that enables it to divert packets to the AE based on their IP/UDP headers. The new generation of high-performance IP routers has this option implemented as part of the router hardware [12]. Edge routers that handle low-bandwidth links may perform this function in software, as demonstrated in our prototype.
- Active manager — The core of the AE is the active manager. This module generates the sessions, coordinates data transfer to and from sessions, and cleans up after a session when it terminates. While a session is alive, the active manager monitors session resource usage, and can decide to terminate its operation if it consumes too much resources (CPU time or bandwidth) or tries to violate its action permissions.
- Security stream module — This module resides in kernel space below the IP output routine. Every connection 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, transparent to the application developer. The module is not fully implemented yet.
- Router interface — This module allows sessions to access the router MIB. It is implemented as a Java object that communicates with the router using SNMP. We are currently working on enhancing performance by caching popular MIB objects.

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

Next, we describe the flow of packets through the system. Note that a nonactive packet does not pass through the AE, 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 the unique session id of the packet, a content description (data, language), and more (Fig. 2). All the diverted packets are sent to the active manager. If a packet does not belong to an existing session and contains code, it triggers creation of a session. If it is a data packet, it is discarded and an Internet Control Messages Protocol (ICMP)-like error packet may be sent. A session creation involves, among other things, 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.

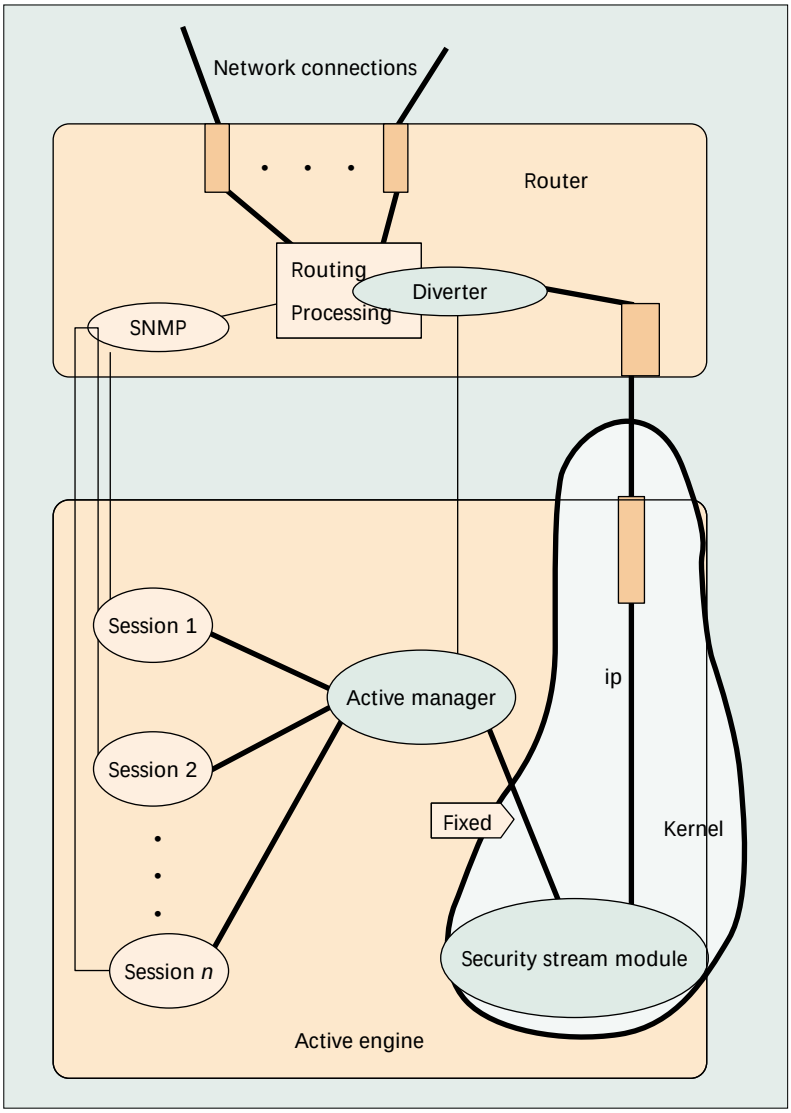

Figure 1. *The general architecture. Thick lines between components represent possible flows of data, thin lines logical connections.*

Four UDP port numbers (3322–5) are assigned to active network research. In our system we use the first two in the following way. The first, the *blind* addressing port, is used to send active packets to unspecified nodes in a certain direction (i.e., toward some distant destination). The diverter in the first active node on the route to that destination intercepts the packet and sends it to the AE. Therefore, 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 it reaches its destination.

Since we expect most network programming to be stable, we do not try to optimize the capsule model. Thus, we are less concerned about program size since programs are not going to be transmitted frequently. A mechanism to reassemble a program from a chain of up to 256 UDP packets is currently implemented in the AE.

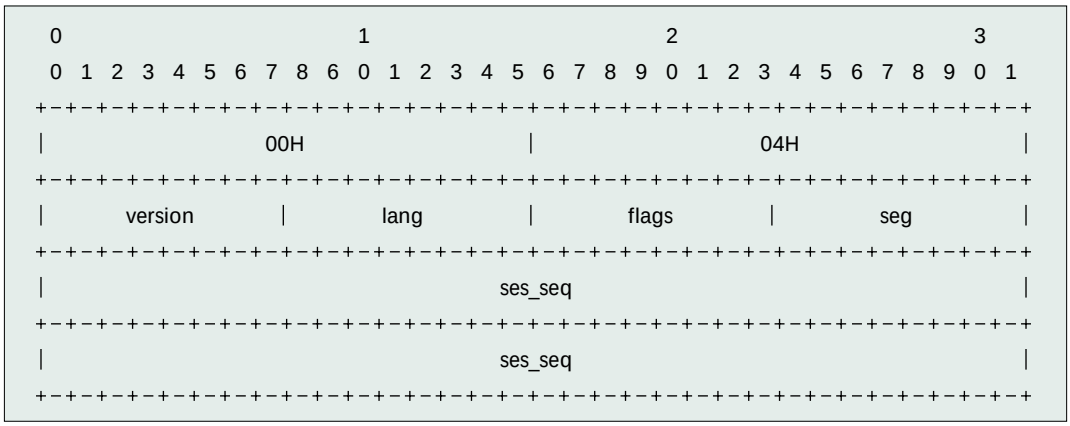

Figure 2. *The structure of the default option in our ANEP header.* `version` *is the software version (currently 1);* lang *is the language id, 0 identify data, 1 is for Java; only the most significant bit in the* `flags` *field is currently assigned to identify the last segment;* `seg` *is the segment number.* `ses_seq` *and* `ses_id` *constitute the session id.*

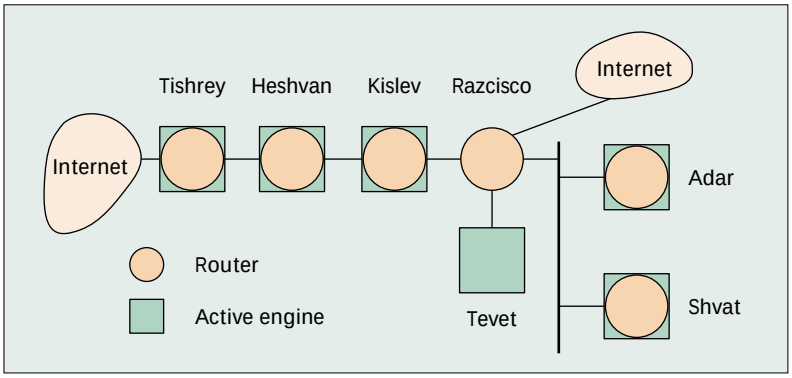

Figure 3. *The prototype network architecture.*

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 AE as a whole should not effect the routing of nonactive 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 nonactive 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 nonactive packets) and resources (CPU time, bandwidth, memory). Since it is important to ensure both safety and security in order to promote the use of active networks, 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 nonactive packets are not affected by active packets. This is easily achieved by the logical separation of the AE from the router. Where the AE is a separate machine and the separation is physical not only logical, which we foresee as the natural evolution path of IP networks, even a crash of the AE will not affect nonactive data.

The next step in safety is to ensure that a session will not corrupt or even poke at other session 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 overuse of system resources is of great concern. To this end, we intend to monitor the use of CPU time by sessions. We implemented tight control over usage of the communication channel to the outside world. TCP connections can be opened only 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 AE. An unauthorized connection will be dropped. UDP packets can be sent only through the manager, which again can monitor the bandwidth usage.

PERFORMANCE

We built a small heterogeneous network comprising both FreeBSD-based active routers and COTS routers (currently we use CISCO 2500 routers and Lucent Technologies RABU PortMaster3) with an adjunct active component. The FreeBSD routers are PCs running FreeBSD, using routed for routing. In these PCs, the AE and router coreside in the same machine. Figure 3 shows the topology used for performance measurements.

In building the prototype we aimed for functionality, not performance. Thus, many parts of the system were 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 a 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` (Fig. 3) exchange UDP packets using either blind addressing (diverted at heshvan) or explicit addressing (forwarded by heshvan's router). The packets were about 500 bytes long, and the network was kept without additional traffic to prevent queuing delay from affecting the results.

The average round-trip delay (RTD) for a packet without diversion was 1.37 ms. The 90 percent confidence interval is 1.37 ± 0.0047 ms, about 0.68 percent wide. The average RTD for a packet with diversion was 11.20 ms. The 90 percent confidence interval is 11.20 ± 0.022 ms, about 0.39 percent wide. Thus, the average delay through heshvan's active engine was (11.2 – 1.37)2 = 4.915 ms. Note that heshvan is a Pentium machine with 64 Mbytes memory running at 200 MHz.

It is obvious that for network management applications the AE performance is bounded by the memory access speed, not by the computation power of the processor. We did not conduct a full-scale stress test, but repeated the above experiment with ten sessions active in heshvan, and the delay through the AE did not change.

APPLICATIONS

Traceroute is an important network management application. We describe several implementations of a generalization of the traceroute program in [13]. Our programs can collect any available network information along a route via SNMP. In addition, the program can be started at any node, not necessarily on the route, and the reports can be sent to any host in the network. Different implementations aim to optimize different criteria, such as minimizing termination time or bandwidth overhead. A full detailed description of the implementation and performance analysis, as well as descriptions of other applications, such as bottleneck detection and ad hoc message dissemination, appear in [13].

DISCUSSION AND FUTURE WORK

Since the inception of the active network idea there has been a search for the "killer application," the one that will strongly require active network technology. We believe network management is a domain where active network technology could prove to be very significant. Applications like adaptive control, router configuration, element detection, network mapping, and security management (intruder detection, fighting denial-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, and more.

The additional delay seen by packets in active networks is an issue that has been addressed in the past, and as others have pointed out, we believe the small slow-down is compensated for by the big savings that can be achieved in traffic volume. In our prototype, nonactive packets suffer only the negligible additional delay of the diverter, and a reasonable delay at the AE. The design and analysis of algorithms for such an environment is an important problem. We took a first step in this direction in [14].

REFERENCES

[1] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation," *2nd IFIP/IEEE Int'l. Symp. Integrated Network Mgmt.*, Washington, DC, Apr. 1991, pp. 95–107.

[2] S. Mazumdar, "Inter-Domain Management between CORBA and SNMP," *7th IFIP/IEEE Int'l. Wksp. Dist. Sys.: Ops. and Mgmt.*, L'Aquila, Italy, Oct. 1996.

[3] J. Kiniry and D. Zimmerman, "A Hands-On Look at Java Mobile Agents," *IEEE Internet Comp.*, vol. 1, no. 4, July/Aug. 1997, pp. 21–30.

[4] M. Zapf *et al.*, "Decentralized SNMP Management with Mobile Agents," *6th IFIP/IEEE Int'l. Symp. Integrated Network Mgmt.*, Boston, MA, May 1999.

[5] G. Hjalmtysson and A. Jain, "Agent-Based Approach to Service Management - Towards Service Independent Network Architecture," *5th IFIP/IEEE Int'l. Symp. Integrated Network Mgmt.*, San Diego, CA, May 1997, pp. 715–29.

[6] K. Psounis, "Active Networks: Applications, Security, Safety, and Architectures," *IEEE Commun. Surveys*, vol. 2, no. 1, 1st qtr. 1999, http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html

[7] B. Schwartz *et al.*, "Smart Packets for Active Networks," *OPENARCH '99*, Mar. 1999, pp. 90–97.

[8] E. Amir, S. McCanne, and R. Katz, "An Active Service Framework and Its Application to Real-Time Multimedia Transcoding," *SIGCOMM '98*, Sept. 1998, pp. 178–89.

[9] S. Bhattacharjee, K. Calvert, and E. W. Zegura, "An Architecture for Active Networking," *HPN '97*, Apr. 1997.

[10] Y. Yemini and S. da Silva, "Towards Programmable Networks," *Wksp. Dist. Sys. Ops. and Mgmt.*, Oct. 1996.

[11] D. S. Alexander *et al.*, "The Active Network Encapsulation Protocol (ANEP)," http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt, 1997.

[12] V. P. Kumar, T. V. Lakshman, and D. Stiliadis, "Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet," *IEEE Commun. Mag.*, vol. 36, no. 5, May 1998, pp. 152–64.

[13] D. Raz and Y. Shavitt, "An Active Network Approach to Efficient Network Management," Tech. rep. 99-25, DIMACS, May 1999, http://dimacs.rutgers.edu/TechnicalReports/1999/99-25.ps.gz

[14] D. Raz and Y. Shavitt, New Models and Algorithms for Programmable Networks, Tech. rep. ITD-99-38382S, Lucent Technologies, Nov. 1999.

BIOGRAPHIES

DANNY RAZ (raz@lucent.com) received his doctoral degree from the Weizmann Institute of Science, Israel, in 1995. From 1995 to 1997 he was a post-doctoral fellow at the International Computer Science Institute (ICSI), Berkeley, CA, and a visiting lecturer at the University of California, Berkeley. Since October 1997 he has been with the Network and Service Management Research Department at Bell Laboratories, Lucent Technologies. His primary research interest is the theory and application of management-related problems in IP networks.

YUVAL SHAVITT (shavitt@lucent.com) received a B.Sc. in computer engineering (cum laude), an M.Sc. in electrical engineering, and a D.Sc. from the Technion — Israel Institute of Technology, Haifa, in 1986, 1992, and 1996, respectively. From 1986 to 1991 he served in the Israel Defense Forces, first as a system engineer and the last two years as a software engineering team leader. After graduation he spent a year as a postdoctoral fellow at the Department of Computer Science, Johns Hopkins University, Baltimore, Maryland. Since 1997 he has been a member of technical staff in the Network and Service Management Research Department at Bell Laboratories, Lucent Technologies, Holmdel, New Jersey. His recent research focuses on active networks and their use in network management, QoS routing and partitioning, and location problems.

Since the inception of the active network idea there has been a search for the "killer application," the one that will strongly require active network technology. We believe that network management is a domain where active network technology could prove to be very significant.