Active Networks for Efficient Distributed Network Management

Danny Raz *Yuval Shavitt*[*]
{raz, shavitt}@lucent.com
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733-3030

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 queueing and fast IP filtering are incorporated into the router hardware. The management of a network comprised of such devices and efficient use of the new functionality introduces new challenges. A truly distributed network management system is an attractive candidate to address these challenges.

In this paper, 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, that enables the safe execution and rapid deployment of new distributed management applications in the network layer. This system can be gradually 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.

Keywords: distributed management, active networks, network management.

[*]Corresponding author Tel: +1 732 949 2303, Fax: +1 732 949 0399

1 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 and take corrective actions by sending configuration commands to network entities.

There are many drawbacks to a centralized architecture, that become more evident with the growth in network size and complexity. As the number of controlled elements grow – 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 the recent years. Most of these solutions delegate some of the central management tasks to distributed software agents or to remote objects. The use of distributed object paradigms abstracts the implementation details. Abstractions, such as CORBA, DCOM, and 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 to be, in many cases, inefficient in their use of network resources, primarily in their use of bandwidth. Another drawback of agent based system 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 for network management calls for the *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 in. Clearly, this has a major impact on the performance since it effects 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 active network as a framework which allows the distribution and execution of network management applications in the network routers.

In our architecture a node is comprised of 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 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 can duplicate themselves. Agents belonging to the same session in different nodes can cooperate their work by exchanging data messages. Using a well-defined network layer interface, namely a 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 by using a well defined API between them. Our architecture uses several security and safety mechanisms that are discussed in Section 4.3.

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 they wish to communicate with. This may be due the the size of the network, or to changes in the network topology. Naming systems such as the ones used by CORBA can help in 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 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 the algorithms performance. Physically, the forwarding mechanism and the active engine may either reside on different machines or coreside inside the same box. This structure enables the upgrade of any COTS (commercial off the shelf) IP router to a node in our architecture simply by adding an adjunct active engine. This presents a natural evolution path from todays 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 comprised of 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 paper is organized as follows. Section 2 gives a short overview of the system we built. In Section 3 we discuss related work both in the fields of network management and active networks. Section 4 describes the system architecture and flow of information. Section 5 demonstrates the architecture capabilities with two implementation examples. We discuss future work and give our concluding remarks in section 6.

2 System Overview

In this section, we provide a general overview of our architecture. For convenience we use the 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 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 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 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. 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 non-active) environments, etc. We also support the explicit addressing mode in which a packet is sent to a specific active node.

3 Related Work

The first generation of distributed network management systems used delegation to alleviate the load from the management station, by allowing the manager to run processes in remote locations [13]. 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 [7]. 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) [5] 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. [14]. They allowed 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 et al. [4]. They built a system where installed agents can manipulate data streams in a router, and suggested a new router design. Though 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 [8] aimed at replacing 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 gradualy enhance them with mature active network techniques. Similar goals were seeked in the BBN Smart Packet project [11] that uses small one-packet scripts for management applications. This project is restricted by the small maximum program size and the lack of soft state maintainance.

In our design, we separate the active engine from the router forwarding mechanism. A similar approach was recently reported by Amir et al. [2], however, they are limiting the scope of their active server to the application level and thus limiting its capabilities. Bhattacharjee et al. [3] 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.

Interestingly, SNMP escaped the notice of most the active network projects but a few (e.g., [12]). In general, most of the research on active networks [8] 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.

4 Architecture

4.1 Design principles

Our goal is to design a system that allows the easy and fast deployment of distributed network management applications. 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 both long- and short-living applications.

Modularity We separate the 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 [1], 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 as a natural evolution 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.

Network layer interface It is 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. This access should support read and write operations since 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, 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 Adding new functionality should not affect the legacy network operation. Further more, 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 to our needs. This technology allows a flexible execution environment for the management applications, without the need to define new protocols and standards. This resulted in the architecture desacribed in Section 2. This simple modular structure supports inter-operability, and does not require that the specific address of the next active hop is known. This structure allows an easy incremental deployment in heterogeneous networks, and is also robust since non-active traffic could not be effected by active traffic. A significant entity in our design is the session. 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, 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 in the following subsections.

4.2 Detailed design

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 if based on their IP/UDP header. The new generation of high-performance IP routers [6] has this option implemented as part of the router hardware. Edge routers that handle low bandwidth links may perform this function in software, as we demonstrate in our prototype.

Active manager — The core of the Active Engine is the Active Manager. This module 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. 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 the details of the Java

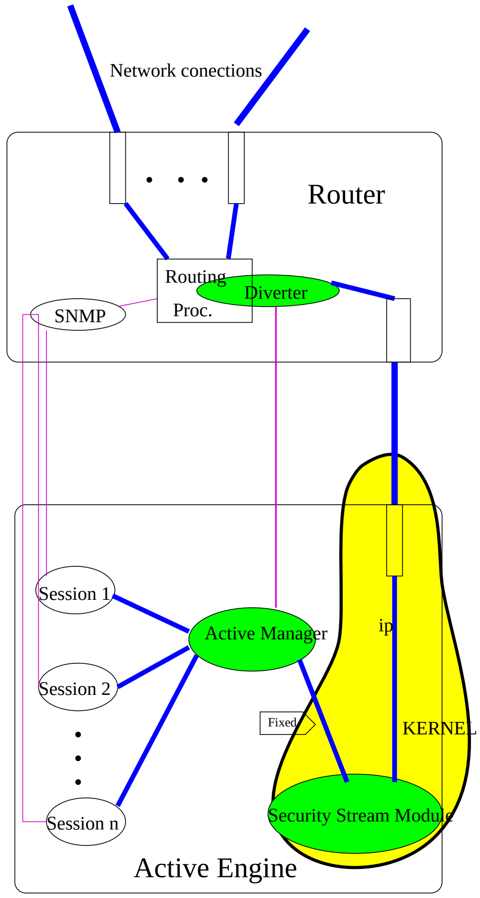

Figure 1: The general architecture. Thick lines between components represent a possible flow of data, thin lines represent a logical connection.

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 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 the unique session id of the packet, a content description (data, language), and more (see figure 2). 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 and an ICMP-like error packet may be sent. 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.

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., towards some distant destination. The diverter in

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              00H              |              04H              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |     lang      |     flags     |      seg      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            ses_seq                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            ses_id                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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` comprises the session id.

the first active node that is on the route to that destination intercepts the packet and sends it to the active engine. 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 reaching 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 the program size as 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 active engine.

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 select initially 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 separation of the active engine from the router. In the case where the active engine is a separated, i.e., 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 active engine will not effect non-active data.

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 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 Performance

We built a small heterogeneous comprised of 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 active engine and the router co-reside in the same machine. Figure 3 shows the topology used for performance measurements.

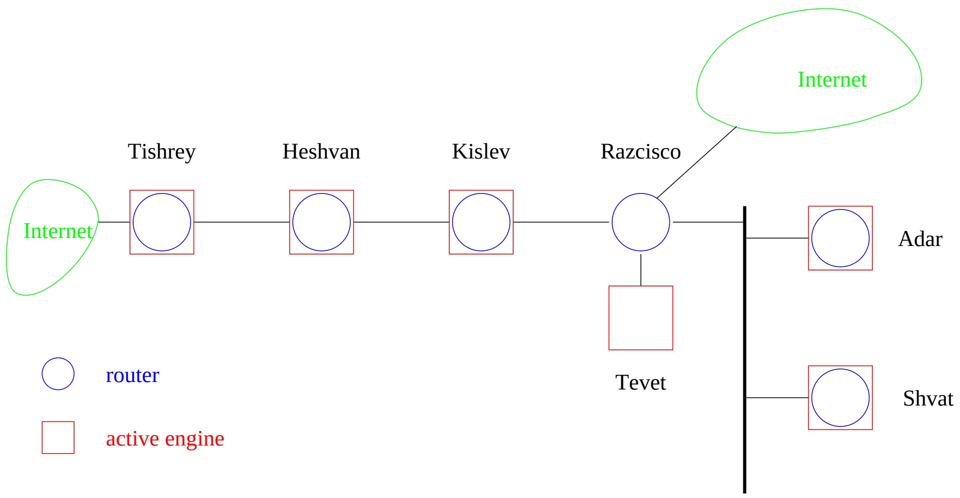

Figure 3: The prototype network architecture.

In building the prototype we aimed at functionality and not at 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 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 3) exchange UDP packets using either blind addressing (diverted at `heshvan`), or an explicit addressing (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 without diversion was 1.37ms. The 90% confidence interval is 1.37±0.0047ms, about 0.68% wide. The average RTD for a packet with diversion was 11.20ms. The 90% confidence interval is 11.20±0.022ms, about 0.39% 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 64Mbytes memory, that runs at 200MHz.

It is obvious that for network management applications the active engine performance is bounded by the memory access speed and not by the computation power of the processor. We did not conducted 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 Applications

Traceroute is an important network management application. In [9] we describe several implementations of a generelization of the traceroute program. Our programs can collect along a route any network information that is available 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. The different implementations aims at optimizing different criteria, such as minimizing the termination time or minimizing the bandwidth overhead. A full detailed description of the implementation, performance analysis, as well as description of other applications, such as bottleneck detection and ad-hoc message dissemination appears at [9].

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 prove to be very significant. Applications like adaptive control, router configuration, element detection, network mapping, 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 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. In our prototype, non-active 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 enviroment is an important problem. We took first step in this direction in [10].

References

[1] D. Scott Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromytis, Gary J. Minden, and David Wetherall. The active network encapsulation protocol (ANEP). URL `http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt`, 1997.

[2] Elan Amir, Steven McCanne, and Randy Katz. An active service framework and its application to real-time multimedia transcoding. In *SIGCOMM'98*, pages 178–189, September 1998.

[3] Samrat Bhattacharjee, Ken Calvert, and Ellen W. Zegura. An architecture for active networking. In *HPN'97*, April 1997.

[4] Gísli Hjálmtýsson and Ajay Jain. Agent-based approach to service management - towards service independent network architecture. In *fifth IFIP/IEEE International Symposium on Integrated Network Management - IM'97*, pages 715 – 729, May 1997. San Diego, CA, USA.

[5] Joseph Kiniry and Daniel Zimmerman. A hands-on look at java mobile agents. *IEEE Internet Computing*, 1(4):21–30, July/August 1997.

[6] Vijay P. Kumar, T. V. Lakshman, and Dimitrios Stiliadis. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, 36(5):152–164, May 1998.

[7] Subrata Mazumdar. Inter-Domain Management between CORBA and SNMP. In *The 7thIFIP/IEEE International Workshop on Distributed Systems: Operations & Management,*, L'Aquila, Italy, October 1996. IEEE Computer Society.

[8] Konstantinos Psounis. Active networks: Applications, security, safety, and architectures. *IEEE Communications Surveys*, 2(1), First Quarter 1999. http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html.

[9] Danny Raz and Yuval Shavitt. An active network approach to efficient network management. Technical Report 99-25, DIMACS, May 1999. http://dimacs.rutgers.edu/TechnicalReports/1999/99-25.ps.gz.

[10] Danny Raz and Yuval Shavitt. New models and algorithms for programmable networks. Technical Report ITD-99-38382S, Lucent Technologies, November 1999.

[11] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell, and C. Partridge. Smart packets for active networks. In *OPENARCH'99*, pages 90–97, March 1999.

[12] Yechiam Yemini and Sushil da Silva. Towards programmable networks. In *Workshop on Distributed Systems Operations and Management*, October 1996.

[13] Yechiam Yemini, Germán Goldszmidt, and Shaula Yemini. Network Management by Delegation. In *The Second IFIP/IEEE International Symposium on Integrated Network Management*, pages 95–107, Washington, DC, April 1991.

[14] Michael Zapf, Klaus Herrmann, Kurt Geihs, and Johann Wolfgang. Decentralised snmp management with mobile agents. In *sixth IFIP/IEEE International Symposium on Integrated Network Management - IM'99*, May 1999. Boston, MA, USA.