Towards Efficient Distributed Network Management

D. Raz *Y. Shavitt*
raz@research.bell-labs.com, shavitt@ieee.org
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 scalable management of a network comprised of such devices and efficient use of the new functionality introduces new challenges. A promising approach is to distribute the network management applications and execute them with minimal central control.

This work concentrates on the way multiple distributed control tasks can be deployed in IP networks. Using a prototype that uses active network techniques we show how truly distributed applications can be used for control and monitoring. We study basic management applications, show the potential gain from running them distributively, and demonstrate their implementation.

Keywords: active networks, distributed network management, IP networks

1 Introduction

The emerging next generation of routers exhibit both high performance and rich functionality, such as support for virtual private networks and QoS [7, 10]. To achieve this, per flow queueing and fast IP filtering are incorporated into the router's hardware [10, 11]. The management of a network comprised of such devices and efficient use of the new functionality introduces new challenges.

Network management applications are traditionally centralized around some manager. The manager queries the managed objects, collects element alerts, builds a view of the network, and informs the operator if a problem is detected. The manager can also try and take corrective actions by sending configuration commands to network entities. However, due to the increase in network size there is a wide spread tendency to alleviate the load from the central management station by delegating some of the work load to agents [3, 1]. Agents are software entities running on a remote machine and perform management tasks on behalf of the central manager. They can be as simple as SNMP agents which serve only as an interface to a remote machine MIB, but may also include a certain level of intelligence, e.g., [3].

While the agent approach relieves the load from the central station, it does not answer other problems associated with central control in large networks, such as: long control loops, increased cost of management, and increased complexity. Thus, the ultimate goal of network management should be a fully distributed control mechanism, where software running on the major controlled devices, e.g., routers, performs most of the network management tasks, leaving very few tasks, such as visualization and operator interface, to a central station. Namely, most of the network management tasks will be done by software running on or next to the network elements with peer relationships, i.e., not as part of a management hierarchy (although an hierarchy can be built ad hoc as part of a specific application).

A few works that suggested this approach before, most notable are the observations made by Mountzia [13]. She identified the fundamental problems associated with using agent technologies for distributed network management. However, the details of her design, in particular the interaction with current information models, are left open. Furthermore, there is no specific example of network management application that can benefit from her work. There are suggestion to use mobile agents for distributed network management. However, most works on mobile agents [5, 8] concentrate on application level, and thus usually neglect the need to interface with the network layer information base (MIB). An exception of this is a recent work by Zapf et al. [18] that allows an application level agent to communicate with a local SNMP agent via a special local service code.

The trend in distributed and delegated network management architectures is to rely on multiple levels of abstraction via the use of distributed object paradigms such as CORBA, Java RMI, and DCOM. While these abstraction mechanisms prove very helpful for high level control they are not suitable for control of layer 3 and 4 functions since they rely on services from these layers. Moreover, these mechanisms may introduce inefficiency since they do not expose the cost (in network resources and time) of atomic operations to the programmer. As is well put in the last chapter of [17]:

> "When CORBA is used the wrong way, the implemented applications, although they are functionally complete, can have performance and scalability problems."

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).

Our approach to network management calls for the distribution of the management tasks in the network. It 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. This work was recently followed by the work of Stadler et al. [6, 12]. A similar approach of using active network technology was taken by Schwartz et al. [15] in their 'smart packets' project. However, their implementation is based on dedicated languages and their programs must fit into a single packet.

This work concentrates on the implementation of a truly distributed network management framework and in particular on basic applications that will benefit from it. We examine the information model requirements, i.e., how can network layer information can be accessed efficiently by the management modules, and how this information can be shared across the network. We demonstrate our approach on a prototype [14] based on active networks techniques. In order to make the discussion clear we concentrate here on basic network management applications. The same concepts and ideas apply also to much more complex distributed network management applications like the congestion avoidance solution described in [9].

The rest of the paper is organized as follows. Section 2 gives a short overview of the prototype we built to demonstrate the applications. Section 3 describes how distributed applications are implemented with that system. We then describe, in depth, two applications: bottleneck detection in

Section 4 and message dissemination in Section 5. We discuss future work and give our concluding remarks in section 6.

2 System Overview

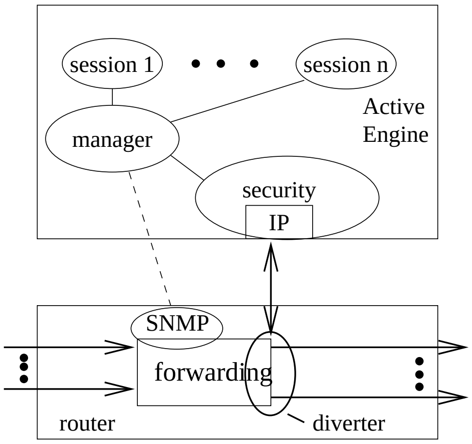

Figure 1: The general architecture.

In this section, we provide a general overview of the prototype system we built in Bell Labs using active network techniques. Full details are available in [14]. Active networks is a framework where network elements, primarily routers and switches, are programmable [16]. 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.

A node in our system is comprised of two entities (see Figure 1): 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 active engine is an environment, where user written programs can be executed with close interaction to the router data and control variables[1]. The separation protects the IP traffic from the effects of an erroneous operation of the active engine, and inflict minimal additional delay on data traffic. It also makes gradual deployment in current networks an easy task, since any COTS IP router can be upgraded simply by adding an adjunct AE.

The IP router performs the IP forwarding and basic routing. It also performs IP filtering that enables the diversion of control packets (or other packets specified by an authorized application) to the active engine. It also provides network layer information to the control applications through an SNMP interface. In addition, a vendor specific mechanism is used to control the data flow, and, optionally, to alternatively retrieve management data.

The active engine is an environment in which code contained 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. When code associated with a non-existing session arrives, it is executed and creates a process that handles all the packets of

[1]The active engine can be perceived as a an execution environment in the context of [4].

that session. Such a process can either handle only a single data packet and then terminate (capsule), or it can reside in the AE for a long period of time handling many data packets as required by many network management applications.

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.

Overall, the system 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.

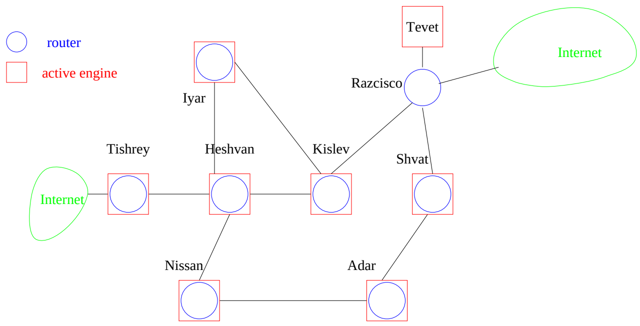

Figure 2: Testbed description.

3 Distributed task implementation

The system described above support multiple simultaneous distributed tasks. All the processes, messages, and data associated with a particular distributed task comprise a session, that has a globally unique id number. This session id is used by the local managers in the AE to demultiplex messages to the appropriate processes.

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 standard 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).

Recently, we introduced a new information retrieval mechanism in the AE to be shared by all sessions [9]. This mechanism allows sessions to share commonly used local information (e.g., neighbor list, local machine id, etc.) and thus amortizes the cost of retrieving it.

There are several ways to instantiate a session in a node. A session can be pre-installed at nodes at system start-up. However, only basic services are expected to be pre-installed, e.g., the squares in Figure 3 represent a basic monitoring session that periodically perform sanity tests. For the rest of the sessions we implemented an active mechanism where the code is send inband. For example, the triangles in Figure 3 represent a session that is spreading in parts of the network. The message dissemination application of Section 5 behaves this way. Finally, sessions can follow a migration path and move between nodes. The bottleneck detection example from Section 4 is a typical example of such a mobile application, which we depicted with circles in Figure 3.

In order to perform distributed tasks, a distributed process at a node must have means to communicate with peers belonging to the same session in other nodes. Relying on the fact that the full topology information is available at every node does not scale. To tackle this problem we support a topology-blind addressing mode that enables a process to send a packet to the nearest AE 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 node.

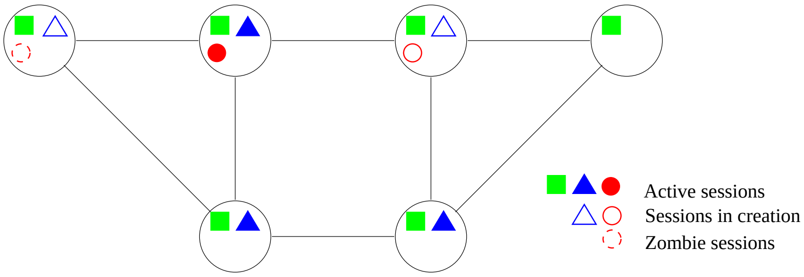

Figure 3: An depiction of multiple distributed sessions

To get a better grasp of the possible performance of such a management system, we briefly report here some of the measurement we performed on ABLE. Note that in building the ABLE 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 were not optimized for performance. Thus on a commercial system one may expect significantly better performance. All our measurements were done on a Pentium machine with 64Mbytes RAM running at 200Mz.

The overhead of intercepting a packet by the diverter, sending it to a session in the AE, and forwarding it back to the IP router is less than 5mS. Accessing a single SNMP object using a FreeBSD agent took 5-6mS in the majority of the cases but could take twice as much in rare cases due to agent scheduling problems, accessing a large SNMP table took much longer. However, using our new cache mechanism [9] the accessing time for all data objects decreased significantly, singular object retrieving time was less that 2mS and large tables were almost always retrieved in 10mS or less.

Our system can handle very large messages by segmentation and reassembly at the AE level. The typical size of simple network management application code that we implemented varied between 2KBytes for the simple bottleneck detection applications in Figure 7, to 12KBytes for the rather complex congestion avoidance application reported in [9]. The data these applications exchange is typically much smaller and usually below 400 bytes.

4 Bottleneck detection

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. We will describe this application in greater detail to give the reader a better understanding on how the system can be used to optimize different parameters.

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 (see example is Figure 4). 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.

```
tishrey# traceroute 135.180.142.33
traceroute to 135.180.142.33 (135.180.142.33), 30 hops max, 40 byte packets
 1  heshvan (135.180.142.2)  0.547 ms  0.398 ms  0.375 ms
 2  kislev (135.180.142.10)  0.588 ms  0.544 ms  0.526 ms
 3  razcisco (135.180.142.18)  2.193 ms  2.192 ms  2.159 ms
 4  shvat (135.180.142.33)  2.588 ms  2.541 ms  2.447 ms
```

Figure 4: An example of a `traceroute` execution from host `tishrey`.

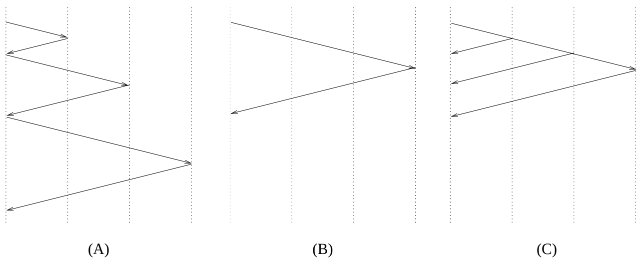

Figure 5: Three traceroute executions on a three-hop path: (A) the current program; (B) collect-en-route; and (C) report-en-route

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 function. One option (collect-en-route) is (see figure 5(B)) to send a single packet that will traverse the route and collect 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 5(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.

The efficiency of these algorithms is measured both in terms of time complexity and communication complexity. Time complexity is the expect time for completing a task, the communication complexity is the overall number of messages (or bytes) that are sent during the algorithm execution. It is easy to see that the `traceroute` program has (see figure 5(A)) time and communication complexities that are quadratic in the path length. Table 1 compares the complexities the three options.

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).

The use of general programs in the capsule enables the application programmer to query any available variable (e.g., a MIB variable) from the router. Our solutions can collect any desired datum 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 in order to identify the bottleneck link.

In addition, the program can be generalized to allow a node to perform the data collection on the path between any other two active nodes in the network. This is facilitated by our two addressing modes. As mentioned before, the reports can be sent to any host (not necessarily active).

```
hop 1: FreeBSD tishrey.dnrc.bell-labs.com 3.2-RELEASE FreeBSD
3.2-RELEASE #0: Thu Jul i386

hop 2: FreeBSD heshvan.dnrc.bell-labs.com 3.2-RELEASE FreeBSD
3.2-RELEASE #0: Fri Jul i386

hop 3: FreeBSD kislev.dnrc.bell-labs.com 3.2-RELEASE FreeBSD 3.2-RELEASE
#0: Thu Jul i386

hop 4: Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-I-L), Version 11.3(3), RELEASE SOFTWARE
(fc1)
Copyright (c) 1986-1998 by Cisco Systems, Inc.
Compiled Mon 20-Apr-98 18:23 by phanguye

hop 5: FreeBSD shvat.dnrc.bell-labs.com 3.2-RELEASE FreeBSD 3.2-RELEASE
#0: Thu Jul i386
```

Figure 6: An example of a router id report generated by our program.

Figure 6 shows the router id report generated by the implementation of option report-en-route, executed on the network depicted in figure 2. The active packet that generates this report can be sent from any host as long as its path goes through an active node. The first active node[2] (`tishrey` in our case) diverts it to its active engine, as the packet uses the well known active port number (3322). The packet contains the class file of the Java code we show in Figure 7 as well as 9 bytes of

[2]A node with an AE.

data, which contain the report destination IP address, the IP address of the destination end-point of the path, and a hop count.

As the session number of this packet does not match any existing session in this node, a new session will be created using the Java code in the active packet. The packet itself is then delivered to this session as the first packet. The session reads the data from the capsule, generates a copy of the active packet to be sent towards a destination, sends a report home, and terminates. The generated copy is then intercepted by the next node on the route to the destination in which exactly the same scenario repeats. The reports are sent to the destination specified in the code (it is easy to have the report destination as part of the data carried in the capsule), which may as well be different from the host that originated the application.

The Java code of Fig 7 is straight forward; `session` is a new instance of the class Act, the constructor takes -9 as an argument that indicates the number of data bytes in the capsule. The program and the data are then retrieved using our Act class methods. A new active packet with the appropriate hop count is then prepared, and sent to the destination address. We then generate a report; local information from the router is gathered using the SNMP interface. Currently, we require here a full MIB specification of the requested values. In the future, part of this interface may be overridden by a different Java interface to retrieve some of the most important information.

Our implementation is not limited to collecting only node ids. We could, for example, check some of the IP counters in a router, instead of (or in addition to) its name. The change in the code is minimal: all we have to do is to request a different MIB variable. If we chose to request `.1.3.6.1.2.1.4.6.0` which is a MIB variable counting the number of IP packets forwarded by the router we get a report like the one in Figure 8. Note that reports are received at `inbar` which is not an active node.

In Figure 8, one can see that the number of forwarded packets is increased between the two executions. The counter value did not change for the last router, since it did not forward any packets between the two executions. Note that the reports may arrive out of order due to the difference in response time between the SNMP servers in the machines. It is the application's GUI responsibility to display the information in a convenient way to the user.

It is easy to generalize the program to start the data collection from any active node in the network. This can be done by using the explicit port number (see page 5) to send the program to the desired starting node. As explained before, the reports can be sent to any host (not necessarily active)[3]. Using this, a reverse traceroute can be easily implemented.

As our generalized data collection program will become popular it can be assigned a well-known session id and be flooded to all the routers. At this stage, small data packets will be sent to collect the data between two end nodes. Such data packets can contain the IP address of the start and end nodes for the path, the report destination, the required MIB variable(s), and a hop count.

5 Message dissemination

In many network management applications there is a need to deliver a message to an ad-hoc group of machines. For example, using an autoconfiguration application, a group of routers might need to be reconfigured due to a change in the network. A monitoring application may periodically query

[3]To reserve this option, we relinquished the ability to display the message travel time that is calculated by the `traceroute` program. However, it can be easily done by the option report-en-route program in the cases where the originator is the node that receives the reports.

```java
import Act.*;
import OurSnmp.*;
public class capsule
{
    public static void main ( String args[] ) throws Exception {

        DatagramPacket  udppacket;
        Act   session = new Act(-9);
        byte[] p = session.getProg();
        byte[] v = session.getInitVars();
        byte[] destip = new byte[4];
        byte[] udpmsg;

        //  get target IP address
        for (int i=0;i<4;i++) destip[i] = v[i+4];

        //  get hop number
        int  hopnum = (int) v[8];
        if (v[8]>127) System.out.println("too big.........");
        else v[8]++;

        //  prepare a new message
        byte[] newpck = new byte[p.length+9];
        for (int i=0;i<p.length;i++) newpck[i] = p[i];
        for (int i=0;i<9;i++) newpck[i+p.length] = v[i];

        // send a new message forward
        session.send(newpck,Act.IPaddr(destip));

        // get some local status (via SNMP)
        String oid = ".1.3.6.1.2.1.1.5.0";  //host name
        String res1 = OurSnmp.Get(oid);

        // send a UDP datagram to report your status
        String udpmsgtext = "hop " + v[8] + ": " +  res1;
        session.sendUDP(udpmsgtext, "inbar.dnrc.bell-labs.com", ReportPortNum);

        // be nice, report you are done.
        session.killme();
    }
}
```

Figure 7: The Java code in the active packet that implements the data collection along a path (option report-en-route)

```
hop 1: 5155
hop 2: 3866
hop 3: 1275
hop 4: 820
hop 5: 396

hop 1: 5216
hop 2: 3922
hop 3: 1288
hop 4: 834
hop 5: 396
```

Figure 8: An example of a report generated by our active data collection program for the IP forwarding counter.

all the hosts it did not hear from in the last period. A security application might collect information from a group of routers based on the attack pattern it suspects.

In these applications, the machine group is ad-hoc defined for the purpose of a single message dissemination and not a long lasting group as in multicast applications. Since the group is defined by the recipient list of a single message, it is not efficient to form a multicast group or to invest in any other long term infrastructure. Without active network, the two ways to implement a message dissemination to a large group of receivers is by either sending a unicast message to each receiver, or by broadcasting the message to the entire network.

We assume that a message is comprised of a header with a list of receivers, and a body which, for a large group of receivers, is much smaller than the header. We use the fact that the union of all the routes from the originator to the receivers is a directed tree rooted at the originator, termed *the dissemination tree*. To simplify the discussion, we assume this tree is a binary balanced tree with the receivers at the leaves (see Figure 9).

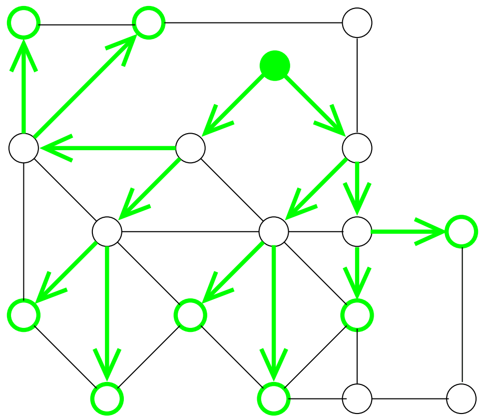

Figure 9: An example of a binary balanced dissemination tree.

Our solution is to partition the receiver list at the source according to the first hop on the path to each receiver. We continue this partitioning at every intermediate node until the message arrives at the tree leaves. This way, exactly one copy of the message traverses each link in the dissemination tree. For a balanced binary tree with n leaves, our message complexity $2n$ and the bit complexity is $O(n \log n)$, while the unicast solution has a message complexity of $n \log n$ and the bit complexity is $O(n \log n)$. The amount of savings achieved by our algorithm in a network, in general, depends on the dissemination tree topology. As a general rule of thumb, if a link has many descendants in the dissemination tree it will contribute more to the efficiency of our algorithm.

In each active node, we have to partition a possibly large list of addresses. This requires more processing at each node than the one required in the bottleneck detection example. In fact, the processing time will be linear in the receiver list length, since we have to check the next hop of every receiver. To be able to do that, we must have access to the routing table at the router which is also a feature we supply through the MIB access. Such an access is typically not available to higher layer software agents.

The overall processing cycles needed in the entire network to accomplish this task for a full binary tree with n leaves is $O(n \log n)$. If the message is distributed using unicast, we only need to have $O(n)$ processing cycles at the sender. The delay due to the processing is about twice in our active solution than the unicast solution. This is since at every level of the tree the processing delay is halved.

Our solution for the message distribution problem demonstrates how active networks techniques can be used to trade-off delay and network utilization. Note, that we achieve a logarithmic improvement in the utilization by paying only a constant factor in delay.

6 Discussion and Future Work

In this paper, 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.

The application examples discussed in this paper demonstrate some of the strength of our approach. However, in order to fully realize our approach power one needs to look at more complex applications. Consider, for example, the reactive monitoring problem discussed in [2] where one needs to identify whether the sum of some MIB variables in a *group* of network elements exceeds a given threshold. Using our approach, a distribute application can be built in a way where the peer instances in the managed devices communicate among themselves locally, and report to the central management station only when the threshold is very likely about to be violated.

Another application that can benefit from the distributed approach is the congestion avoidance application mentioned before. In this application, routers in a close locale coordinate corrective action in order to divert traffic from congested links to less congested bypass routes [9].

References

[1] N.G. Aneroussis and A.A. Lazar. An architecture for controlling service demand in atm networks based on pricing agents. In *Workshop on Distributed Systems Operations and Management*, October 1996.

[2] Mark Dilman and Danny Raz. Efficient reactive monitoring. In *Infocom 2001*, Anchorage, AK, USA, April 2001.

[3] Germán Goldszmidt and Yechiam Yemini. Distributed management by delegation. In *International Conference on Disributed Computing Systems*, June 1995.

[4] 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.

[5] Günter Karjoth, Danny B. Lange, and Mitsuru Oshima. A security model for aglets. *IEEE Internet Computing*, 1(4):68–77, July/August 1997.

[6] Ryutaro Kawamura and Rolf Stadler. Active distributed management for ip networks. *IEEE Communications Magazine*, 38(4):114–120, April 2000.

[7] S. Keshav and Rosen Sharma. Issues and trends in router design. *IEEE Communications Magazine*, 36(5), May 1998.

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

[9] Jessica Kornblum, Danny Raz, and Yuval Shavitt. The active process interaction with its environment. *Computer Networks*, April 2001. To Appear. AN early version appeared in IWAN'01, Tokyo, Japan.

[10] 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.

[11] T. V. Lakshman and D. Stiliadis. High speed policy-based packet forwarding using efficient multi-dimentional range matching. In *SIGCOMM'98*, September 1998.

[12] K. S. Lim and R. Stadler. A navigation pattern for scalable internet management. In *IEEE IM 2001*, Seattle, WA, USA, March 2001.

[13] Maria-Athina Mountzia. A distributed management approach based on flexible agents. *Interoperable Communication Networks*, 1(1):99–120, January 1998.

[14] Danny Raz and Yuval Shavitt. Active networks for efficient distributed network management. *IEEE Communications Magazine*, 38(3), March 2000.

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

[16] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[17] Andreas Vogel and Keith Duddy. *JAVA Programming with CORBA*. Wiley, 2nd edition, 1998.

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