Environments for Active Networks

Rosen Sharma sharma@cs.cornell.edu
S. Keshav skeshav@cs.cornell.edu
Michael Wu mwu@cs.cornell.edu
Linda Wu lxwu@cs.cornell.edu
Department of Computer Science
Cornell University
Ithaca, New York, 14853

Abstract

Active networks have the potential to solve three fundamental problems in traditional (and current) networks: (a) *service extensibility*, (b) *rapid reaction* to network events, and (c) obtaining a *disaggregated* view of network state. Traditional networks expose a fixed set of network services, which are difficult to modify or customize. In contrast, active networks allow service extensibility by providing an environment for execution of nearly arbitrary agents (or applets) *within* the network. Such a framework can be used to build application level gateways for multimedia processing, such as transcoding and fusion/merging, within the network

A second advantage of active networks is their ability to rapidly respond to asynchronous network events because control algorithms can be co-located with network elements. Finally, traditional networks achieve scaling by aggressively aggregating network state. Applets embedded in the network see disaggregated state, allowing finer-grained control.

In this paper, we describe some preliminary ideas in exploiting an active network framework for next-generation networks. We fix ideas by describing our experience in using an active network framework for network management.

1 Introduction

We define active networks as networks that allow programs called *agents* or *applets* to be executed in internal network nodes such as multiplexors, firewalls, routers and switch controllers. Rudimentary active network components are already used for mobile communication (for forwarding packets after a handoff), in video conferencing (for prioritized dropping of multicast groups), and firewalls (for allowing customizable filters).

To clarify the behavior of a realistic active network component, consider Figure 1, which shows the architecture of an active firewall. Such a firewall prevents "undesirable" packets from entering the protected subnet. Packets are classified using a script, such as the one shown in Figure 2.

Scripts used in this way are very similar to applets. Such scripts enable new application data streams to be filtered through the firewall. For example, to permit an application to receive data on one of two UDP ports through the firewall, only a small script needs to be written. This demonstrates the *extensibility* property of active networks.

Note that the script assumes that the applet has access to packet headers. If this access is not available, the script cannot work. In other words, the execution environment of the applet must include a way to read packet headers. In general, rich execution environments need to be defined to enable the creation of powerful applets.

This example also contrasts a *declarative* framework for network control with a *procedural* or active framework. In a declarative approach, a host would declare a protocol type and source from which to allow (or disallow) packets. This is intrinsically limited to a predefined set of protocol and source descriptions. In contrast, the procedural framework outlined here allows the creation of customized application level libraries, which can be arbitrarily customized. Although extensibility comes at the price of some inefficiency, this can be avoided by selectively subsuming procedural functionality in a declarative framework.

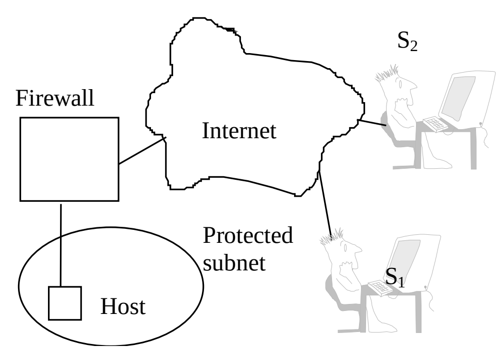

Figure 1 Firewall

```
if (pkt.protocol == UDP){
   if(pkt.src == s1)
      allow the packet
   else
      drop the packet
}
upload next script
```

Figure 2: A sample script that runs on the firewall in Figure 1.

Consider the scenario shown in Figure 1, and assume that the link between the host and the firewall is a slow link (such as a wireless or modem link). Suppose the host is participating in a videoconference with two other participants S_1 and S_2, but only has bandwidth to receive the stream from one of the sources. Moreover, the host wants to switch between the sources rapidly. This could be achieved if an applet running on the firewall subscribed to both the sources, but forwarded packets from only one of the sources. The applet can intelligently decide which stream to forward depending not only on the source's wishes, but also on the current state of the network. For example, assume that the host switches from S_2 to S_1, but due to congestion in the network packets from S_1 arrive only after some delay. If the applet notices that there are packets from S_2 buffered in the firewall, it could forward packets from S_2 in order to utilize the bandwidth of the slow link between the firewall and the host. This illustrates the use of applets for near-real time control, where an applet running at the point of control can react to network state faster than an endpoint.

To illustrate the disaggregation property of applets, consider an applet running at the firewall that discovers congestion on the path from S_1. It may then decide to find an alternate path to S_1. If the firewall and the subnet's egress router are co-located, the applet can find this information from the router's network topology database without requiring the host to acquire this information. The host only needs to set its "default" route to the router, and the applet takes care of the rest. The default route represents aggregated routing state stored at the endpoint. In contrast, the applet has access to disaggregated state to make fine-grained control decisions.

We now show how to use applets to disaggregate state even further, at the cost of additional bandwidth. Note that a typical routing protocol will update the network topology database at the egress router only periodically, around once every 30 seconds. If the source wanted to discover network state at a finer time-scale, it could install applets on several alternate paths. Each applet could communicate with the applet at the egress router to inform it of its current load. The applet on the egress router could choose one of the paths dynamically (using source routing) based on this information. This represents the use of applets simultaneously to disaggregate state on provide faster response time to network state.

This extended example shows how an active network framework allows us to design networks that are extensible, exhibit rapid reaction to network events, and have access to disaggregated information. We can exploit such a framework to design new protocols, extend existing ones, or improve the performance of existing systems. We will also discuss the use of an active network framework for network management.

This paper is organized as follows. Section 3 discusses the key benefits of an active network framework. Sections 4, 5, and 6 discuss instances of protocols or systems where these core advantages are exploited. Section 7 discusses our solution to the problem of defining object models. Section 8 presents some problems we encountered in using this framework.

2 Key Advantages of an Active Network Framework

The many advantages of an applet-based framework have been thoroughly discussed in the literature [7, 5]. Here, we enumerate only those advantages that we have identified to be useful in building next-generation networks.

2.1 Extensibility

Traditional networks expose a fixed set of network services, which are hard to modify and to customize. In contrast, active networks seek to create an extensible system by providing an environment where nearly arbitrary applets can be executed, allowing new services to be dynamically created. Such a framework allows us to rapidly deploy and refine protocols, dramatically reducing the development cycle. As discussed in Section 2, in contrast to declarative frameworks which need to encompass the needs a broad range of applications and yet be compact to be useable, a procedural framework allows the creation of customized application level libraries, some of which maybe later made part of the declarative framework. An applet-based framework also facilitates the mechanisms for developing the glue between different networks and protocols.

2.2 Disaggregation

2.2.1 Semantic filtering

Endpoints often need to make decisions based on the current network state. There are two ways in which applets help an endpoint obtain this information. First, network state information may not be available at the desired granularity. For example, a host might be interested in the routing policies in the subnet of its transport-level peer, but the backbone routing protocol may tell it only about the routing policy in the backbone network, since information about subnets is lost during the aggregation process. Second, a host may not interested in raw information but may want to process the information before using it. If the host obtains unprocessed (and uncompressed) information from the network, this wastes bandwidth. In both situations, an applet sent into the network can obtain the information at the granularity required by the host, additionally filtering out the information not needed by the source.

2.2.2 Time scale of information gathering

In order to increase stability of the network, dynamic information about the network is propagated at a very slow time scale (on the order of tens of seconds). Increasing the time scale over which information is averaged reduces the overhead of transmitting the information and also helps in aggregating control information over time. But different sources might want information over different time scales. Traditional architectures do not allow this. By installing applets inside the network, endpoints can obtain information at the time scale that they require.

2.2.3 Horizon effect

As networks grow larger, the overhead of distributing information increases. Hierarchy reduces overhead in large networks by grouping nodes into clusters. Only aggregated information is available about the cluster outside the cluster. This introduces a problem that we call the *horizon effect*, where a host can not obtain information about nodes in a remote cluster. An applet that is sent into the network can overcome the horizon effect by directly accessing disaggregated information.

2.3 Rapid response to network events

There are some unavoidable latencies associated with transferring information over the network, even if it is filtered and sent at the time scale that the end system desires. This latency can be a hindrance if this information is used to control entities over the network. By co-locating control algorithms with the control element, one can reduce the latency of the control operation, thus provide accurate and rapid control.

In the following sections we see how the three general principles stated above can be used to (a) add flexibility to existing architectures (Section 4), (b) extend protocols and systems (Section 5), and (c) improve the performance of existing networks (Section 6).

3 Adding flexibility to existing architectures

3.1 Protocol stacks for firewalls and mobility

As mentioned in Section 2, a firewall allows applets to access the packets belonging to a flow. Firewalls can be implemented at several layers of the protocol stack. The higher the level at which the firewall is implemented, the greater the access to semantic information (such as port numbers and protocol types) but the lower the efficiency. For example a firewall can be easily implemented in user space at the cost of copying data into and out of the kernel. Or, it can be implemented in the transport layer where the inside and outside connections share buffers, thus avoiding extra packet copies, but ignoring application-level semantics. By exposing all the layers of the protocol stack to the firewall applet, we can migrate firewall functionality from one layer to another, dynamically trading semantic information for efficiency.

A parallel problem arises in the use of applets in mobile TCP schemes. The mobile station can send an applet to the base station to improve the performance of TCP connections. This applet can run at the network layer to give the abstraction of a reliable link; at the transport layer to exploit intelligent caching mechanisms; or at the application layer to forward packets after a handoff by splitting a TCP connection into two connections.

Exposing the entry and exit points of each layer of a protocol stack allows us to unify the above individual solutions into one general framework, which allows

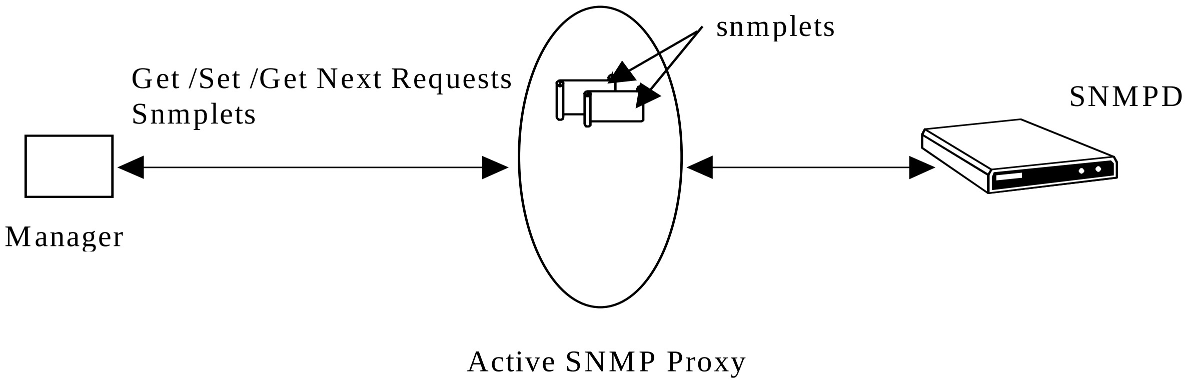

Figure 3 Overview of the Architecture

mobile schemes or firewalls to be implemented at any layer of the protocol stack. The framework also allows applets to enhance functionality, for example provide QoS, by using other exposed models like those of scheduling and buffer management.

3.2 Multi-Mode Communication

Providing qualities of service that are customary in the phone networks on the Internet seems to be a daunting if at all feasible task. The networks differ both in the service quality they provide and the type of traffic they carry. The Internet provides best effort delivery and whereas the telephone network provides guaranteed service. Voice traffic is constant bit rate and is well understood whereas data traffic on the Internet is not well understood, ever changing in character and bursty on all time scales. Given these fundamental differences, we believe that different networks will coexist at least in the near future. Indeed, we believe that there are many interesting applications that can creatively exploit services provided by both networks.

Consider, for example, an audio conferencing application that supports both audio and a whiteboard. For audio we may want to use the phone network, while we want to use the Internet for the shared whiteboard. This is an example of what we call *multi-mode* communication. It tries to leverage the strengths of each network and provide interoperability in both the data and control planes. Exposing object models for switching elements such as IP routers and telephone switches to intelligent applets provides a framework for developing multi- mode applications. To continue with our example, the audio conferencing application could contact an applet running on a PBX to establish a bridged (multicast) voice connection and another applet running on an MBONE-capable router to set up a shared whiteboard connection.

Note that both extensions described in this section do not need any modification to existing protocols. In contrast, in the next section, we discuss extensions to existing protocols to better exploit an active network framework.

4 Extending current protocols and systems

4.1 SNMP

The designers of SNMP [24, 25] purposely made it simple because they did not expect managed entities to have adequate computing power. Today, however, managed entities run operating systems (like pSoS) which supports Java applets and are powerful enough to do things beyond the scope of SNMP. To take advantage of the increased computing power available in managed entities, we enhanced SNMP by allowing applets to be sent to the managed entity, enabling real time control policies, fine-grained measurement, and sophisticated trap generation algorithms. (This also allows “semantic routing”, where applets move from one endpoint to another based on some information present on the host. For example, an applet could use SNMP to find a path with the most number of RSVP compliant routers between a source and a destination.)

We now describe a framework that we developed and tested for enabling Active SNMP, an extension of the widely used SNMP network management protocol. The architecture of our system is shown in **Figure 3**. The three main components are the manager, the proxy, and the managed entity. The proxy interacts with the managed entity on behalf of the manager. It allows us to convert an existing managed object to an “active” network element *without* modifying it in any way. Managers send standard SNMP requests to the proxy, which simply forwards them to the managed entity. However, managers can also send *snmplets* (pronounced “snaplets”) to the proxy. The snmplets are executed in the proxy and have access to the SNMP Management Information Base (MIB) at the managed object.

In SNMP, the manager has to poll the managed entity to obtain information about dynamically changing variables in the MIB [24]. The possibility of a large latency between the manger and the managed object precludes the implementation of fine grained control algorithms. Also in periods of congestion SNMP requests may be lost. A control applet running in the proxy overcomes these drawbacks as it has direct access to the variables in the MIB. Moreover, because it is co-located with the managed entity it is able to control it even in periods of network congestion.

To communicate with a snmplet, a host needs to name instances of snmplets running in a given proxy. We identify snmplets with a two-part name. The first part is an HTTP URL, and the second part is a domain-specific name (this is similar to the way classes are named in Java). We embed snmplet names and parameters in existing SNMP MIBs. Each instance of a snmplet is accessed through a variable in the MIB that acts as a resource handle. We also exploit the MIB to store intermediate snmplet state. When the snmplet starts or moves to another machine it may need some parameters or startup information. This information is also kept and accessed through the MIB. For example, the originator of the applet sets variables in the MIB to pass parameters to the applet. The structure of the information in the MIB is described in ASN.1 similar to all other information stored in these modules. This unification of snmplet state and the MIB leads to a uniform and simple access of persistent state information.

We developed an active proxy for an IP router and exposed MIB-II [22] and the IP forwarding MIB [2] to applets running on the router. This allows snmplets to access the routing and forwarding tables on the router. One of the first snmplets we wrote monitored the ICMP unreachable messages that were sent by the router and informed the managing workstation when there was a sharp increase in these messages, indicating that one of the neighboring routers was down.

The framework allowed us to easily develop and deploy other snmplets. Another snmplet we developed ran on the router and collected statistics on packet arrival. The applet varied its averaging interval based on the amount of traffic on the network, taking more samples when the network was under high load. In a traditional framework the management station would have to continuously poll the router to obtain this information and would tend to take less samples during periods of overload as it does not want to further congest the network. Thus the active proxy allows us to obtain more accurate and useful information.

We have also developed a scheme that allows proxies to verify the identity and the authenticity[1] of snmplets. In terms of performance, the active proxy does increase the latency to access information by a small amount, but we believe the increase in functionality to be worth this overhead.

4.2 Signaling (RSVP)

Signaling protocols like UNI or RSVP are declarative in that they only specify the quality of service a connection requires and not how to obtain it. Efforts to make these protocols expressive, in order to deal with a broad spectrum of quality requirements, leads to an exponential increase in the number of parameters. Moreover, the class of applications that may use these protocols is unknown. We can avoid these inherent problems if signaling is carried out by applets that access a well-understood interface at each router and carry out reservations for the endpoints. Eventually, on gaining sufficient understanding of how to model the environment we intend to come up with a declarative signaling protocol that is likely to be a more compact and usable version of the existing protocols. Our approach here, therefore, is to build on top of the object models developed for router, then define models for scheduling and routing.

5 Applets for improving network performance

In this section, we present some scenarios where we can use an active network framework to improve the performance of current networks.

5.1 Adaptive routing

Routing today is usually based on computing the shortest path between a source and a destination. Using shortest path routing requires the use of a *single* best path to a destination. In the presence of congestion on this path, it is desirable to use other paths in the network. This is usually termed multi-path routing.

Proposals for multi-path routing can be broadly classified depending on whether the routing decision is made at each hop (hop-by-hop routing) or whether the decision is made at the source (source routing). In hop-by-hop routing, coordination among the nodes using alternate routes is necessary in order to prevent routing loops. The alternative to hop-by-hop routing is source

[1] Our local area network was behind a firewall, thus applets had permissions to read and write the MIBs.

routing. In source routing the source of the packet determines the route the packet will take to the destination. Unlike hop-by-hop routing, where each node must make consistent routing decisions to prevent loops in the topology, source routing is able to deal with inconsistent state in the network

However, the use of source routing requires information about the network to be available at each endpoint in the network. This introduces an inherent tension in the routing protocol. Widespread dissemination of routing state incurs high overhead, introduces instability, and questions the timeliness of the information available to a source. Breslau [6] has evaluated schemes like Hop Count Limited Flooding, Random flooding and Reverse Path Update to trade off performance of routing with the amount of overhead propagated in the network. For example, Hop Count Limited Flooding (HCLF) restricts the propagation of information to the desired number of hops. Using HCLF with a radius of 1 (3) achieves 23.4% (66.2%) of the benefits of flooding, while its overhead is about 3% (12%) of that of flooding.

An applet installed in the network can either compute alternate paths in a distributed manner, or, at the cost of increased bandwidth, send filtered information back to the source. It can also accommodate source-specific policies while choosing alternate paths. Thus, by allowing applets to run within the network, we can release the tension between widespread dissemination of routing information and the use of source routing to find alternate paths in the network.

5.2 Fast Renegotiation

Applets can also be used to support fast renegotiation during call setup. When the reservation fails at a router in the network an endpoint might want to repeatedly try to obtain this reservation. This introduces unnecessary overhead in the network[2]. An applet that is located on the router can act on behalf of the source and periodically transmit messages when the state of the reservation changes or explore alternate paths along which future reservations might succeed.

[2] This is similar to the "killer" reservation problem in RSVP, where, due to the merging of reservations, an end-point that repeatedly tries to reserve a large amount of resources in the network can lead to denial of service to other end points whose reservation, if not merged with this large reservation, might succeed.

6 MIBS as Execution Environments

As mentioned in Section 1, rich execution environments need to be defined to enable the creation of powerful scripts or applets. At each router the applet is presented with an execution environment. To define new environments at each router and have router vendors implement them is an uphill and time consuming task.

Our extensions to simple network management protocol (SNMP) allow applets to access information from the MIB. Variables in the MIB have a well-defined environment, whose semantics and naming conventions are standardized. This allows us to better understand the potential of applets, without incurring a huge startup cost of defining environments.

Although the MIB may restrict the interfaces available to an applet, this is an intermediate solution which allows the development and testing of several new application level protocols. Ultimately if this proves to be a useful and powerful architecture we will be able to export interfaces to components like routing and signaling directly without going through SNMP.

7 Problems

An active framework adds flexibility or improves the performance of existing architectures, but it introduces some unique problems.

7.1 Aggregation

In a procedural (active) framework, the network lacks the intelligence to aggregate information. For example, applets injected by different sources may be collecting the same information. How can we aggregate this information to avoid sending the same information twice over the network? Two techniques for this purpose are described in Reference [29]. First, as described in Section 5.1, each applet and its instance could be uniquely identified. Thus, an endpoint could discover that an instance of an applet that it wants to run is already present on an intermediate node. Second, some applets may be defined as filters to transform the results of one applet into another. For example, two applets collecting the same information but over different time scales, can be replaced by the applet collecting information at a finer time scale and an applet which averages the information as the second applet would have done.

7.2 Security

Security and performance are concerns in the implementation of the active network architecture. Programming languages like Java provide security at the level of individual applets, but do not allow multiple protection domains (many applets) to exist within a single address space. For example web browsers do not provide secure communications between two applets on the same page. Work in operating systems [4, 9, 23] has explored the use of safe languages to allow untrusted code (thus many protection domains) to run inside the kernel (a single address space). Recent advances in language technology like the Safe Language Kernel [10] allows different applets to execute in a secure and efficient manner in the same address space. We believe that these advances adequately answer the inherent security concerns in an active network architecture.

8 Related Work

Our vision of active networks is inspired by the seminal work by Tennenhouse et al described in References [26, 27, 1]. In their framework, applications inject "capsules" rather than packets into the network. A capsule is a packet where the header identifies what piece of code should be run for the packet. Our work proposes to use applets only in the *control path*, thus avoiding the performance penalty incurred by using applets in the data path.

References [7, 12, 11, 28] provide a high-level survey of the advantages and disadvantages of using applets. They conclude that applets offer significant advantages in cases of disconnected operation, individualized service, server side execution, high bandwidth-low latency interactions, but they also introduces problems like performance limitations, secure environments, transmission efficiency.

Other work focuses on different aspects of the applet framework. References [29, 13] describe the ARPA Knowledge sharing effort which led to the definition of an applet communication language (ACL), which consists of a vocabulary, Knowledge Interchange Format (KIF) and the Knowledge Manipulation and Query Language (KQML). ACL is a declarative language (similar to SQL), that represents knowledge as tupules and first order logic. Reference [5] classifies frameworks based on their capability to support applet migration. This is similar to the JavaBeans [14] effort for Java. References [16, 15, 17, 19] describe various aspects of Java and the Java virtual machine, which is the language in which applets are written.

NOW [1] and SPIN [4] describe operating systems that allow user code to be executed in a safe manner in the kernel. This allows user applications to directly access hardware like MPEG boards or to customize the drivers.

9 Conclusion

We believe that active networks allow us to explore different approaches to solve several existing problems. Our work has applicability beyond active networks as it helps us to gain a better understanding of some fundamental issues in networking. Our focus is to identify specific problems where an active network framework can be employed, and to expose environments in those frameworks for the development of useful applets.

Although this framework is very general, it can be customized to support new multimedia applications. Such applications can help to perform transformation, mixing and selective dropping of video, audio and image data in the network. For example, in a videoconference an applet inside a congested network can selectively drop video packets corresponding to the lost audio clips.

Based on the arguments presented in this paper and our experience with an active framework for SNMP, we believe that our approach will allow us to make significant advances in signaling, routing, network management and in enabling multi-mode communication.

References

[1] T. E. Anderson, D. E. Culler and D. A. Patterson, A Case for NOW (Network of Workstations).

[2] F. Baker, IP Forwarding Table MIB, RFC1354, SRI Network Information Center, January 1997.

[3] Fred Baker and J. Krawczyk, RSVP Management Information Base, Internet Draft, November 1996

[4] B.N. Bershad et al, Extensibility, Safety and Performance in the SPIN Operating System. In Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain Colorado.

[5] K. A. Bharat and L. Cardelli, Migratory Applications, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-138.html

[6] L. Breslau, Adaptive Source Routing of Real-Time Traffic in Integrated Service Networks, PhD Dissertation, University of Southern California.

[7] D. M. Chess, C. Harrison and A. Kershenbaum, Mobile Agents: Are they a good idea ?, IBM Research Divsion, T. J. Watson Research Center, Yorktown Heights, NY10598,http://www.research.ibm.com/ massive/ mobag.ps

[8] W. R. Cheswick and S. M. Bellovin, Firewalls and Internet Security, Addison Wesley

[9] E. Cooper, R. Harper and P. Lee. The Fox Project: Advanced Development of Systems Software. Technical Report CMU-CS-91-178, Carnegie Mellon University, August 1991.

[10] Thorsten von Eicken et al, SLK: A Capability System Based on Safe Language Technology, submitted to SOSP 97

[11] L. Fonner, What's an agent, anyway? , http://fonner.www.media.mit.edu/people/foner/Julia/Julia.htm

[12] S. Franklin and A. Graesse, Is it an Agent or just a Program ?: A taxonomy for Autonomous Agents, Institute of Intelligent Systems, University of Memphis http://www.msci.memphis.edu/~franklin/AgentProg.html

[13] M. Genereseth and Ketchpel, Software Agents, http://logic.stanford.edu/sharing/papers/agents.ps

[14] Java(tm) Beans: A Component Architecture for Java http://splash.javasoft.com/beans/WhitePaper.html

[15] The Java Language Environment White Paper, ftp://ftp.javasoft.com/docs/whitepaper.ps.tar.Z

[16] Java Language Overview, ftp://ftp.javasoft.com/docs/java-overview.ps

[17] The Java Platform White Paper ftp://ftp.javasoft.com/docs/JavaPlatform.ps

[18] Java Security FAQ, http://java.sun.com/sfaq

[19] JavaOS(tm): A Standalone Java Environment ftp://ftp.javasoft.com/docs/whitePaper.JavaOS/JavaOS.cover.ps

[20] G. Malkin and F. Baker, "RIP Version 2 MIB Extension", RFC 1724, SRI Network Information Center, November 1994.

[21] McCanne, S., and Jacobson, V., The BSD Packet Filter: A New Architecture for User-level Packet Capture. Proceedings of the 1993 Winter USENIX Technical Conference (San Diego, CA, Jan. 1993)

[22] S. K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC1213, SRI Network Information Center.

[23] H. Mossenbook. Extensibility in the Oberon System. Nordic Journal of Computing 1(1), February 1994.

[24] Marshall T. Rose, The Simple Book: an Introduction to Network Management revised Second Edition, Prentice Hall Series in Innovative Technology, 1996

[25] W. Stallings, Network Management, IEEE Computer Society Press, 1993.

[26] D. L. Tennenhouse, S. J. Garland, L. Shrira and M. F. Kaashoek, From Internet to Activenet , Laboratory for Computer Science, MIT

[27] D. L. Tennenhouse and D. J. Wetherall, Towards an Active Network Architecture. Telemedia, Networks and Systems Group, MIT,

[28] Wooldridge and Jennings, Intelligent Agents: theory and practice, http://www.doc.mmu.ac.uk/STAFF/mike/ker95.ps

[29] M. Wu, R. Sharma and S. Keshav, An Intelligent Framework for Network Management, work in progress.