IEEE
COMMUNICATIONS
SURVEYS

ACTIVE NETWORKS: APPLICATIONS, SECURITY, SAFETY, AND ARCHITECTURES

KONSTANTINOS PSOUNIS
STANFORD UNIVERSITY

ABSTRACT

Active networks represent a new approach to network architecture. Routers can perform computations on user data, while packets can carry programs to be executed on routers and possibly change their state. Currently, the research community is divided concerning the usefulness of active networks. On the one hand, active networks provide a much more flexible network infrastructure, with increased capabilities. On the other hand, they are obviously more complex than traditional networks and raise considerable security issues. The purpose of this article is to provide a broad survey on active networks. The first goal is to highlight their efficiency in a variety of applications. After presenting some key points on each application, we discuss some current experimental technologies and assess the usefulness of active networks in congestion control, multicasting, caching, and network management. The second goal is to address the security issues that active networks raise: the problem is defined, and techniques for solving it are presented and elaborated upon with a description of a specific implementation of a secure environment and related performance measures. Issues related to the design of a programming language for active networks are also discussed. The third goal is to classify active network architectures based on their design approach. Thus an inclusive presentation of currently proposed architectures, which focuses on their design attributes, capabilities, performance, and security, is given.

Traditionally, the function of a network has been to deliver packets from one endpoint to another. There was a distinct boundary between what is done within the network and what is done by the users. Processing within the network was limited basically to routing, congestion control and quality of service (QoS) schemes. This kind of a network can be regarded as "passive". Several problems with "passive" networks have been identified: the difficulty of integrating new technologies and standards into the shared network infrastructure, poor performance due to redundant operations at several protocol layers, and difficulty accommodating new services in the existing architectural model. An additional shortcoming is that, recently, applications which sometimes require computations within the network have emerged, such as firewalls, Web proxies, multicast routers, and mobile proxies. In the absence of architectural support for doing so, these applications have adopted a variety of ad hoc services for performing user-driven computations at nodes within the network. A need was felt to replace the numerous ad hoc approaches to network-based computation, with a generic capability that allows the users to program their networks. This innovative idea of imparting the user the ability to program the network is called active networking.

Active networks represent a new approach to network architecture. These networks are "active" in two ways: routers and switches within the network can perform computations on user data flowing through them; and users can "program" the network, by supplying their own programs to perform these computations [3]. In the extreme case, there will be no difference between internal nodes and end user nodes since both will be able, if needed, to perform the same computations.

The emergence of new technologies supporting encapsulation, transfer, safe and efficient execution, and interposition of programs and program fragments is one of the reasons why it is now possible to build active networks. At the same time, in the fields of operating systems and programming languages, issues relating to mobility, efficiency, and safety have been addressed. From all the above one can conclude that there is a user "pull" and a technology "push" towards a new way of thinking about the network [1, 4]: the user "pull" stems from the paradigms that "violate" the traditional properties of the network while the technology "push" stems from the fact that until recently it was not technologically possible to treat programs as a set of encapsulated and moving code fragments.

The question that arises is whether such a change will improve the performance of the applications that run through a network or not. There is no clear answer to that question

yet. The research community is divided on whether or not active networks are useful. The argument against active networks is that the Internet is successful today because of its simplicity; by making the networks "active" things may get very complicated. The argument for active networks is that it is a very promising and innovative idea; a variety of useful network services that involve processing at intermediate nodes will be made possible and the use of such services is likely to lead to better end-to-end performance for applications [2].

A reasonable way to judge whether or not active networks can improve network performance is via the end-to-end argument. The end-to-end argument is a system design principle intended to help determine where to place services in a subsystem. The argument states that a function or service should be placed "in" the network only if it can be cost-effectively implemented there. The authors in [5] state that some services can best be supported or enhanced using information that is only available inside the network. Since active networks push functions and services "in" the network, these functions and services may then easily, timely, and efficiently use the information available there. Therefore, active networks can significantly enhance the network performance. Two examples that are easy to imagine are time and place of congestion and location of packet losses within multicast distribution trees.

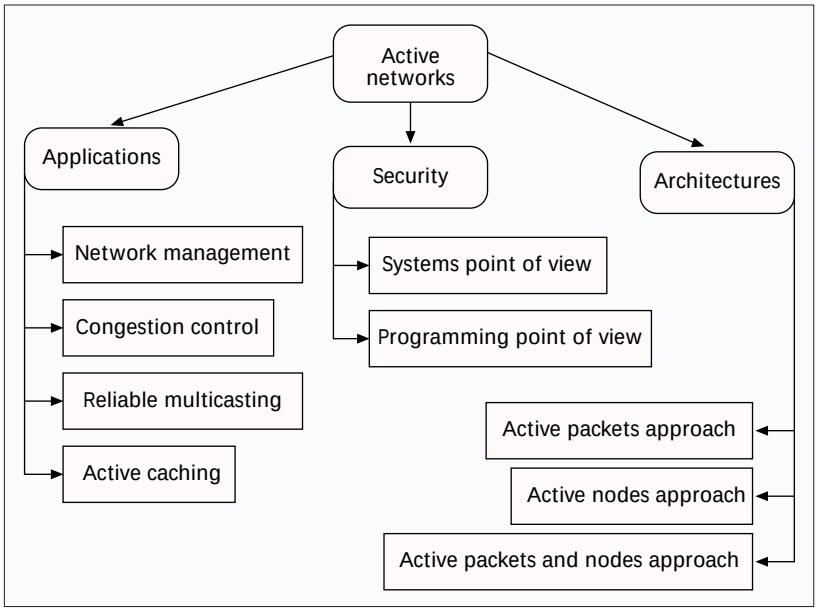

Figure 1. *Structure of the article.*

DEFINITIONS

It is helpful to provide some formal definitions of passive and active networks in order to clarify their differences.

Definition 1 — A passive or traditional network is a network that consists of smart hosts sitting at the edges of the network that are capable of performing computations up to the application layer, and simple routers that interconnect the hosts and can only perform computations up to the network layer.[1]

Definition 2 — An active network is a network that allows intermediate routers to perform computations up to the application layer. In addition, users can program the network by injecting their programs into it. These programs travel inside network packets and are executed in intermediate nodes resulting in the modification of their state and behavior.

In the extreme case, the packets of an active network can be regarded as programs. We call these packets active packets to distinguish them from "traditional" network packets. Conceptualizing a packet as a moving program inevitably brings to mind the notion of intelligent agents and, in particular, of mobile software agents which could be considered a specific class of the former. A formal definition of a mobile software agent may be helpful.

Definition 3 — A mobile software agent is a program that acts on behalf of a user or another program, and is capable of moving within the network under its own control. The agent chooses when and to where it will migrate, and may interrupt its own execution and continue elsewhere on the network. The agent returns results and messages in an asynchronous fashion [7].

The mobile agent paradigm proposes to treat the network as multiple agent-friendly environments and the agents as programmatic entities that move from location to location, performing tasks for users [8]. The similarity between the two ideas is obvious. Indeed, many of the active network Architectures presented later in this article use mobile code techniques that are very close to mobile software agent technology. However, the idea of active networks is much more general. Active networks visualize the network as a collection of active nodes that can perform any computations, and a collection of active packets that carry code and are indeed programs. Under that viewpoint, a mobile agent may be regarded as a specific type of an active packet, and a mobile-agent-compatible node of traditional networks[2] could be regarded as a specific type of an active node since the latter is secure and allows any kind of computations. A fundamental difference between the two ideas is that active networks use the concept of network layer processing whereas mobile agent systems run as application programs. An active network, because it is programmable by each nature, offers the applications of mobile software agents as "primitive functionality."

The structure of this article is as follows: in the next section we present how active networking can be used to improve the performance and the efficiency of specific applications. The first application is network management and three experimental technologies that seek the advantages of active networks over network management are presented, along with general remarks. The second is congestion control, an important case of network management. The third is multicasting, and the fourth is caching, used particularly for Web browsing and multicasting. Experimental technologies are also presented for all these applications. The third section addresses the issue of security that is raised by the increased flexibility of active networks. The problem is analyzed, solu-

[1] *The seven layers of OSI in ascending order are the Physical layer, the Data Link layer, the Network layer, the Transport layer, the Session layer, the Presentation layer, and the Application layer.*

[2] *In a traditional network, mobile agents need special mobile-agent-compatible nodes in order to be executed.*

tions are presented, and an implementation of a secure environment is described together with some measures of its performance. In this section we also address security issues from the programming point of view by presenting some programmability issues and suggesting ways to deal with them. In the fourth section we present the proposed architectures that support active networking. Our conclusions are given in the fifth section. Figure 1 schematically presents the general structure of this article.

APPLICATIONS

The most important application of active networks stems directly from their ability to program the network: new protocols and innovative cost-effective technologies can be easily employed at intermediate nodes. The functions of the nodes will no longer be rigidly built-in by vendors who must follow designs dictated by slow and intractable standards committees. Also, network integrity will not be vulnerable against various ad hoc approaches toward network programming, as is the case today. At the same time, active networks can be very beneficial for a variety of specific applications. In this section we present why and how this is possible for network management, congestion control, multicasting, and caching.

NETWORK MANAGEMENT

Currently, network management is achieved by having management stations routinely poll the managed devices for data, looking for anomalies. This technique has served us well in the past. However, due to the increase in the number and complexity of nodes in the network,[3] now it has become problematic. Management centers become points of implosion, inundated with large amount of information. This information is very often redundant, as the packets that arrive may simply report that there was no change in the state of the monitored part of the network. Also, in case of a problem, the round-trip delay that is needed for the information to reach the management center and the reply to return back to the affected part of the network, is sometimes significant and the action undertaken is not up to date any more. It is essential that network management employs techniques with more immediate access and better ability to scale.

Active networks are the natural answer to the above problem. By making the internal nodes of the network active we can move the management centers right in the "heart" of the network and thus reduce both delays from responses and bandwidth utilization for management purposes. Also, we can inject special code in the packets that can act as "first aid" in case they encounter a problematic node. This code can be executed in the affected node and change its state automatically instead of waiting for a reply from a management center. Other packets can act as "patrols," constantly looking for anomalies as they trace the network. Finally, since a management center sends programs to the managed nodes, it can request real-time tailoring of the information to be returned in order to meet its current needs. This will reduce the back traffic and processing time of the information after it is received by the management center. To sum up, by using active networking for network management:

- Problems are tracked quickly or are reported automatically without the need of polling.
- Management centers can be in the "heart" of the network, thus delays from responses and bandwidth utilization for management purposes are reduced.
- "Patrol" and "first aid" active packets can respectively track a problem and deal with it at once.
- Information content returned to the management centers can be tailored to the current interests of the center so that back traffic and processing time are reduced.
- Management policies can change easily as administrative requirements change, thanks to the inherent flexibility of active networks technology.

Experimental Technologies Related to Network Management — Three projects that apply active network technology concepts to network management will be discussed here as a first look at how the previous points translate in practice. The first is the Smart Packets project [11] held at BBN Technologies, the second is the Network Management by Delegation Paradigm [13, 14] of Columbia University, and the third is the Darwin Project of Carnegie Mellon University [10, 12].

The Smart Packets project is designed to demonstrate the benefits of active networks in network management. Traditional data packets are replaced from smart packets that may carry programs. Smart Packets programs are written in a tightly-encoded, safe language specifically designed to support network management and avoid dangerous constructs and accesses. A closer look at the architecture of the project is provided in the section entitled "Architectures." Smart packets is designed as a network management tool where the manager uses smart packets to efficiently manage network resources. There are four types of packets: program, data, error, and message packets. Program packets carry the code to be executed at the appropriate nodes. Data packets carry the results of the execution back to the originating network management program. Message packets carry information messages and error packets return error conditions. The management of large and complex networks is improved by moving management decision points closer to the node being managed, by targeting specific aspects of the node for information, and by abstracting the management concepts to language constructs.

Network management by delegation is the first application that moved code to be executed in network elements. Its underlying principle is that management processing functions can be delegated dynamically to the network elements and executed locally rather than centrally. Instead of moving data from the managed nodes to the management center, one can move management application code at the network elements where the data resides. Management by delegation provides a powerful computational architecture for scalable, decentralized, and automated management. A distributed technology that supports moving management application code as delegated agents is used to remotely load and execute management software. Delegated agents are a special case of mobile software agents and can be used to deploy functions of arbitrary functionality.[4] An elastic server provides a portable operating system extension that supports the execution of delegated agents. Finally, a delegation protocol is used to dynamically dispatch delegated agents to an executing elastic server at a remote system, and to control their execution. Network management by delegation reduces the network bandwidth utilization for management purposes and the

[3] *The Internet is growing exponentially.*

[4] *Delegated agents are also used to deploy functions in intermediate nodes in the context of Netscript architecture, which is presented in the section on architectures.*

delays due to remote data accessing. In addition, it assists the developers of management applications in modifying their management policies as administrative requirements change. In other words it achieves spatial and temporal distribution of management functionality. Similar work in the context of TMN framework[5] was done at UCL [15].

The Darwin project introduces the term delegate for code that is sent by applications or service providers to network nodes to implement customized management of their data flows. Delegates are executed on designated routers and can affect resource management through a control application programming interface (API). The control API should not be too restrictive because it will limit the usefulness of delegates, but should not give too much freedom either, as this may lead to system inefficiency. The resources are organized in a resource hierarchy in order to be manageable. The functions of the API fall into a number of classes. One class of functions allows the delegate to change the structure of the resource hierarchy by splitting flows, merging flows, changing the resources given to nodes, etc. A second class allows delegates to affect routing. Finally, a third class allows the delegate to send and receive messages. The delegate can monitor the network status either by explicitly requesting information or by asking to be notified about changes. Customization of resource management is a key requirement for high-quality value-added services, and active networks as well as other related tools such as mobile agents provide the framework and technology to make it possible.

CONGESTION CONTROL

Network congestion is a problem unlikely to disappear in the near future. Therefore, it is essential to find efficient algorithms to deal with it. Congestion is a prime candidate for active networking, since it is an intranetwork event, usually far removed from the application. Also, it often takes a considerably long time for congestion notification information to propagate from the point of congestion to the user, so that the latter can self-regulate in order to reduce congestion. As a result, either there is a period of time during which congestion is augmented — since applications have not learned about it — or the notification arrives so late that there is no longer any congestion and self-regulation is not needed. Since congestion control is a special case of network management, all the benefits of active networks presented in the previous subsection are also valid here.

On a descriptive level, one can find many examples where the added functionality of active networks can help in dealing with congestion control. Here are some examples:

- An active node can monitor the available bandwidth and control the rate of a data flow accordingly. Of course, buffering is needed in this case, so instead of putting the buffers in the switch, we can put them in the active node.
- In case of many data flows with different congestion requirements, an active node can control the relevant rate of each flow in addition to the total rate. Also, it is possible to adapt to dynamic changes of the requirements.
- The transformation of data at a congestion point is also a powerful capability. In fact, applications sometimes produce data according to the congestion situation if they are aware of it. Therefore, we can perform the above transformation right in the place where it is needed and only if it improves the performance. However, we should expect that from a computational point of view, a transformation may have a significant cost.
- Selective dropping of units, packets or cells can be held very efficiently. In case of congestion, we prefer to drop less important units than more important ones. The importance of a unit depends on the amount of information it carries. A classic example here is the case of MPEG compressed video where if we lost an I frame, there is no point in keeping the P and B frames that depend on the lost I frame.
- Finally, we can have a multi-stream interaction in the following sense: e.g., if a user is receiving video and audio and there is a loss in the video, audio units should receive extra priority to assure that the user will still get some information.

Experimental Technologies Related to Congestion — The work done at the Georgia Institute of Technology [16] is an example of a current experimental technology that focuses on the benefits of active networking with respect to network congestion. The approach used is that the network defines a finite set of functions, which can be computed at an active network node by the so-called active processor. Also, there is header information in each packet that specifies which computation is to be performed on it, called active processing control information (APCI). Backward compatibility with existing network protocols is achieved because nonactive nodes need not recognize the APCI in order to switch packets, and APCI is not required in packets switched by active nodes.

When a packet arrives at a node, the following steps take place:

1. The output destination port of the packet is computed (as usual).
2. If there is an APCI then the packet is sent to the active processor for further processing. If not, the packet is transmitted.
3. The function specified in APCI is computed.
4. The packet's header and APCI are recomputed if the result of the function is transformed data. Also, the node's state is updated as required by the function.
5. The packet is transmitted.

Obviously, in the worst case, the service provided with active processing should not be worse than the typical service if the node was not active.

The authors of [16] have tested some of the above ideas in an experimental configuration consisting of an ATM switch and four attached Sparcstations, one of which was acting as an active node. The system was heavily congested on purpose. The first experiment tested the performance of the unit-level dropping (ULD) function, according to which units that are meaningful to the application are dropped altogether if any portion of the unit must be dropped. Without active processing, approximately 36 percent of the units were properly received. With ULD, approximately 90 percent of the units were properly received as long as the buffering size was at least four times the application unit size. The second experiment, in which a group of Pictures (GoP) consisted of an I frame and all the B and P frames that rely on it, involved MPEG. It was shown that dropping an entire GoP, in case its I frame is lost, leads to better results than just using plain

[5] *The telecommunications management network (TMN) framework aims to support the management of telecommunications networks and services. Conceptually, the TMN is a separate data network that interfaces to the telecommunications network for purposes of monitoring and control. The TMN relies on the OSI management model for the modelling of network and services resources.*

ULD, particularly if buffer space is limited. Finally, various multi-stream interaction functions were evaluated and compared to each other.

The model presented above may be considered "conservative" because it does not use all the capabilities that active networks offer. The packets only carry the APCI, they do not carry code to be executed in the nodes. Also, the nodes have a fixed set of functions instead of infinite ones. The motivation for such a simple model is that it serves its purpose well without having to change anything in the rest of the network. A partial implementation of the architecture of active networks can offer some direct benefits and help move the network in the direction of more radical changes. More on the architectural considerations of this approach will be presented in the "Architectures" section.

In the future, we expect more radical models to emerge, which will be more powerful but also more complex and less compatible with non-active technology. One idea is to use the per-packet programs to allow each packet to make decisions about how to be routed on the fly. By arriving at a node, a packet may execute its "routing" code according to the information kept in that node. That information may be the result of the processing of information brought to the node by other packets. So, each packet acts as a "monitoring" device of the network and informs each node that traverses about what it had learned during its "journey." Thus, upcoming congestions are tracked and regulation is done automatically before congestion takes place. The above scheme resembles that of a driver who listens to the traffic reports on the car radio and makes on the fly decisions on which roads to use, taking into account his/her own time constraints.

MULTICASTING

The Internet and the next generation of networks will have to handle a great variety of application traffic such as audio, video, teleconferencing, etc. Many of them inherently require multicasting. New techniques are being sought that will provide the functionality of multicasting in an efficient, reliable, and scalable way. Applying the ideas of active networks in this field may prove useful: active internal nodes can elegantly solve many current problems such as NACK implosion,[6] concentrated load of retransmissions, useless retransmissions, duplication of packets and immunity to group membership changes,[7] while existing "passive" schemes provide only partial solutions to the above problems. Indeed, at MIT the inventors of a loss recovery scheme that takes advantage of active networking, called active reliable multicast (ARM) [18], claim that it can solve all the above problems efficiently.

Active Reliable Multicast and Other Research — ARM utilizes intermediate routers to protect the sender and network bandwidth from unnecessary acknowledgments and retransmissions. ARM routers play an active role in loss recovery. They provide soft-state storage and perform customized computation based on different packet types. Storage is soft-state because an active node will cache an item for a specified time as long as it has enough cache capacity. Not all nodes need to be active for the scheme to work properly and efficiently. The percentage of nodes that are active, the size of the caches, and the lifetime of each cached item are obviously important parameters of the scheme. Three error recovery strategies are used by ARM as follows.

- The first action taken by active routers is to cache the data in order to be able to retransmit it in case of a retransmission request. This is called local retransmition. Therefore, as NACKs travel toward the sender, they may trigger the retransmission of a lost packet by an intermediate node. By doing so, both latency is and traffic are reduced. Caching presents a tradeoff between network-based storage and bandwidth. Therefore, it makes sense to cache only at routers located at "strategic" points in the network. Studies on packet loss patterns indicate that most packet losses occur at the edge of the network. Thus, routers that connect stub networks to the rest of the network or routers immediately before lossy links such as wireless links are good candidates for caching. It is interesting to mention that active routers may also cache the repair packets, which allows them to accommodate for possible NACKs from distant areas of the network. Indeed, when storage is limited it is more important to store repair packets than original ones.
- The second action taken by active routers is the process of NACKs aiming at suppressing unnecessary request traffic (NACK suppression) and to get information about the originators of retransmission requests. Each multicast packet includes a NACK count field in its header. Active routers maintain two records: the NACK record and the REPAIR record. The former contains the highest NACK count received for that packet — used for suppressing NACKs — and an indication of the outgoing links on which NACKs have been received — used for scoped retransmission. The latter contains a vector indicating the outgoing links on which a retransmission is on its way, so that future NACKs will be suppressed.
- The third action is the scoped retransmission. When a repair packet arrives, the router checks the NACK record. If no indications exist, it forwards the packet to all outgoing links. However, if there is an indication of the outgoing links that have asked for retransmission, the repair packet is forwarded only to those links achieving scoped retransmission.

Simulation results have shown that ARM has much lower recovery latency than "passive" schemes and provides a specific solution to local recovery. Also, it is flexible and robust because active routers do not have to maintain nor have the knowledge of group topology to efficiently perform the above actions. It is important to say that all the above can be achieved even if less than 50 percent of the routers are active. Two open issues of the above scheme are the location and duration of caching. These issues will be addressed subsequently.

Other research on active networks and multicasting is presented in [17]. The authors have implemented two protocols for multimedia streams and used them to demonstrate the usefulness of active networks. The first protocol, called Robust Multicast Audio, is an example of how the performance and efficiency of an existing protocol can be improved by adding application-specific computational power to internal nodes of the network. The second, called Layered Multicast Video, demonstrates the quick development of a new protocol that optimizes bandwidth usage in multicast trees.

[6] *In case of errors, many receivers send NonAcknowledgement packets towards the sender. The aggregate of that traffic increases as we approach the sender, and may reach the capacity of the nearby links.*

[7] *In multicasting, interested users join a group and a multicasting tree is formed from the sender to these users. Group membership is dynamic in the sence that new users may join the group any time and old users may leave the group any time.*

The above protocols were implemented using two different active architectures, ANTS and M0, which are presented in the "Architectures" section.

Reliable multicasting is still an open issue in the research community. Based on an assessment of what the research has shown so far, it is fair to suggest that the use of active technology is likely to prove even more beneficial in the future as it deals with a key aspect of multicasting: it allows for decisions to be made inside the network, which is an essential necessity for loss recovery.

CACHING

A substantial fraction of network traffic in the Internet comes from applications like the World Wide Web, where information is retrieved by clients from servers located anywhere in the network. The caching of objects at locations close to the clients can decrease both the network traffic and the time needed to retrieve the information. Active networks can be used to provide a smart caching scheme wherein smaller overall storage capacity is needed and higher reduction in network traffic and latency can be achieved. Traditional approaches to network caching place large caches at specific points in the network. The key point in these schemes is how to choose these specific points. One option is to cache at transit[8] nodes (transit-only caching). Since a large fraction of paths in the network have to go through transit nodes, they are prime candidates for caching. Another policy is to cache in stub nodes that are connected to transit nodes because the former have to be traversed in order for a node inside a stub domain to access the rest of the network. Therefore, cache nodes can be located near the edge of the network or at strategic points within the network organized with a hierarchical scheme wherein clients are manually configured to access a particular cache in the hierarchy.

An interesting idea would be to "balance" the hierarchy by repositioning not only the cached information but also the cache nodes. In this scheme, each node or a set of nodes decide whether to cache the information that returns from the server to the client. Obviously, the effective organization of the location and the content of the caches is not trivial. Nodes should be smart enough to cache objects that nearby clients will request in the future and to coordinate with each other to avoid caching the objects that are already cached in neighbor nodes. Active networks technology may help deploy a mechanism of coordinating the nodes. Also, because a significant fraction of Web pages are dynamically computed, active technology may support the storage and execution of programs that generate these pages in nodes near the clients.

Self-Organizing Wide-Area Network Caches — Recent work at the Georgia Institute of Technology considers the benefits of associating caches with nodes throughout the network, and self-organizing cache contents in an active way. The proposed scheme, called Self-Organizing Wide-Area Network Caches [19], yields round-trip latencies that are smaller than or equal to the more traditional approaches, while requiring much smaller caches per node. The basic idea is to obviate the need to decide where to place caches by considering that all nodes of the network can cache objects and relying on active technology to maintain a uniform distribution of caches within the network. Nodes make local decisions in a way that resources are used effectively overall.

The first approach described is called modulo caching. A distance measure, called cache radius, is defined, measured in transmission hops. The caching policy uses the radius as follows: on the path from the server to the requesting client, information is cached in nodes that are cache radius apart. We therefore end up with a distribution of caches located a "cache radius" away from each other. The second approach uses some of the cache space in each node to store locations of information objects. Each node's cache is divided into "levels." Level 0 contains locally cached objects, level 1 contains objects cached in nodes one hop away, etc. When a request message for an object is processed, the levels are searched in sequence beginning with level 0. This approach is called lookaround algorithm. The number of levels of adjacent caches maintained and checked in this algorithm is a parameter of the policy and, as with the cache radius, might be set globally, on a per-object basis, or even locally.

Simulation results show that active mechanisms outperform traditional methods in case of correlated accesses. By correlated accesses, we mean that an initial access will cause future accesses involving the same client and server pair. In case of uncorrelated accesses, transit-only caching performs a little better than active mechanisms, but this sort of caching fails to adapt to correlated accesses.

Active Caching and Reliable Multicasting — The ideas of active caching can be used in the area of reliable multicasting to determine the duration and location of caching dynamically. Active technology can be used to assign values to the lifetime of cached objects according to the nature of the application and the frequency of NACKs at that specific time. By using the current structure of the multicasting tree, nodes may decide on the fly whether they are going to cache packets or not and for how long. In dense areas of the multicasting tree, more caching nodes are needed. In order to accommodate distant receivers, a caching node should hold the packets for longer time. As the tree is formed, active nodes can track the number and distance of their neighbors and make a nearly optimum decision regarding caching. New membership information may change not only the multicasting tree but also the distribution of the cached nodes, resulting in a more "balanced" hierarchy tailored to the current state of the tree.

SECURITY AND SAFETY

So far we have explored various fields of networking where active networks can be useful. In this section we discuss the security and safety issues that active networks raise. Since active networks are much more flexible than passive, the number of safety and security issues that need to be addressed are tremendously increased. By safety we mean reducing the risk of mistakes or unintended behavior. By security we mean the usual concept of protecting privacy, integrity, and availability in the face of malicious attack. A packet that carries executable code can potentially change the state of a node. Nodes (routers, switches, etc.) are public resources and are essential to the proper and correct running of many important systems. Therefore, the safety and security requirements placed upon the computational environment where the code of packets will be executed must be very strict.

[8] *Network can be considered to be a collection of stub and transit domains. Stub domains reside at the edge of the network and are interconnected by transit domains. The former carry traffic addressed to or from some node in the stub domain and the later carry transit traffic.*

In the current Internet, the only resource consumed by a packet at a node is the memory needed to temporarily store it and the CPU cycles necessary to find the correct route. In such an environment, strict resource control in the intermediate nodes was considered non-critical. However, an active packet may consume not only many more resources but also at a faster rate. Denial of service attacks may easily occur if there is no resource management. Clearly, in addition to security and safety, fairness is also an issue.

In an active network, active packets may misuse active nodes, network resources, and other active packets in various ways. Also, active nodes may misuse active packets. Previous work related to the security issues of mobile software agents is directly applicable here [9]. Some of the possible problems that may occur are the following:

- Damage: An active packet can destroy or change the resources or services of a node by reconfiguring, modifying, or erasing them from memory. A node may erase an active packet before the completion of its job in the node. Finally, active packets that share the same computational environment may attack each other.
- Denial of Service: An active packet may overload a resource or service due to constantly consuming network connections or using a great portion of the CPU cycles available. The node cannot function properly under these circumstances and another active packet cannot be executed or forwarded.
- Theft: An active packet may access and steal private information from a node. On the other hand, an active packet is vulnerable toward the node at any point when visiting it. Even if it is encrypted, it is not totally safe because it usually has to be decrypted in order to execute.
- Compound attack: The biggest actual threat for an active node is a compound attack aimed toward a goal. For example, a malicious user may send many active packets toward a central router and try to bring it down by consuming all its bandwidth capacity.

Protecting the nodes and the packets in a flexible environment such as active networks is not an easy task. Some techniques that may be used to protect the active nodes will be presented first, followed by a discussion of ways of protecting the active packets.

- Authentication of Active Packets: Any active packet should have authenticating credentials produced using one of a number of algorithms such as a public key signature algorithm. This do not guarantee that the active packet will be harmless, or even useful. Credentials only provide assurance that someone else vouches for the active packet.
- Monitoring and Control: A reference monitor may be used to restrict the information, system resources and services that active packets are allowed to access and use. The reference monitor consults a security policy to determine if access is to be granted. Since access-level monitoring places restrictions directly on what a packet can do, it is an effective method. However, the decision of granting permission for using some resources is based upon some credentials which are not able to guarantee that a packet is harmless as it is already mentioned.
- Limitation Techniques: Time limits such as the amount of time an active packet may be allowed to be executed, range limits such as the total number of nodes the packet is allowed to traverse, as well as duplication limits (i.e., the number of times that a packet may duplicate itself), are essential in preventing an active packet from monopolizing the resources of a node.
- Proof Carrying Code (PCC): PCC [29] is based on the observation that is often easier to check an answer than to produce it. For a mobile program, it is the creator of the program who knows the key reasons it is correct, not the host (active node) that receives the program. Hence we could pair the mobile program within each active packet with a proof of its correctness. The active node may easily check the proof and then run the program. The difficult part is the creation of the proof but this is the job of the program creator.

Two methods are suggested for the protection of the active packets: fault tolerance techniques and encryption. Encryption refers to the situation where active packets do not consist of cleartext code and data.[9] Encryption is usually used for code and data in transit. However, the programs may even be executed in a non-cleartext form, which leads to the concept of mobile cryptography [26]. The fault tolerance techniques are replication, persistence, and redirection. Replication means that packets replicate at each node. Persistence means that packets are temporarily stored against node failure so that even if a node crashes, the copy persists in storage. Redirection means that packets may seek alternative routes in case their default route fails. Replication and persistence are unacceptable for the vast majority of network packets because they consume memory and bandwidth, and only very "important" active packets should be allowed to do this (e.g., packets installing a new version of a routing protocol in all nodes). Redirection and encryption have broader applications in packet protection because they basically consume CPU cycles. A combination of fault tolerance techniques and encryption may give very good results in the problem of protecting active packets. However, because these techniques are still in their infancy, there is much to be done before definite results are reached.

Combining all of the above,when a packet containing executable code arrives at a node, the system must:

1. Accept the authenticity of the credentials of the packet,
2. Identify the sending network element,
3. Identify the sending user,
4. Authorize access to appropriate resources based on these identifications and credentials,
5. Allow execution based on the authorizations and security policy,
6. Monitor and control access to system resources throughout the execution,
7. If needed, encrypt the packet to protect its code and data in transit.

If the packet is not identified properly, then it may be allowed to execute the code in a restricted environment or it may not be allowed to execute the code at all. There is a form of admission control and policing in the above procedure.

SECURE ACTIVE NETWORK ENVIRONMENT

We present here a Secure active network Environment architecture called SANE [25], along with some measures of its performance [22]. SANE provides a basis for implementing secure active network technologies and has been undertaken by researchers at the University of Pennsylvania.

Security has to be addressed at two levels, static and dynamic. Static concerns are those that only need to be checked infrequently, as in the case of an active network bootstrapping from a cold (idle) start into an operational

[9] *Cleartext data are meaningful to the receiver without deciphering.*

state. Dynamic concerns need to be addressed continuously. The above separation is a guideline for implementing static and dynamic checks. The former can be costly but very secure and the later should be cheap enough so as not to degrade performance. Obviously there is a tradeoff between cost and level of security. In an active network where packets can constantly alter the state of nodes, we expect to have a great deal of dynamic concerns.

Two basic concepts of security are integrity and trust. For an active network node, a trusted layered architecture can be constructed by making lower layers trustworthy and ensuring that higher layers depend on the integrity of the lower ones. Also, a web of trust between participant network elements is required because programs carried in packets travel from one network element to another. Even in that case, a program from a trusted network element may be damaged during transmission. Therefore, it is clear that a combination of static and dynamic checks is required in the case of active networks to ensure security.

SANE is a layered architecture. The lower layers of the architecture ensure that the system starts in an expected state. This is done by using a secure bootstrap architecture called AEGIS [23]. This is a static check and after that, dynamic checks on a per-user or per-packet basis can be made. The higher layers of the architecture are responsible for these checks. The system maintains security in several ways from this point onwards.

- First, it performs remote authentication, when required, for node-to-node authentication.
- Second, it provides a restricted execution environment for the evaluation of the programs received by the network.
- Finally, it uses a novel naming scheme to partition the node's service name space between users.

It is important to mention that every user and every active element own a public/private key pair; these keys are used to authenticate and authorize actions of those entities.

The dynamic integrity checking and other security issues of this architecture have been implemented and tested in the prototype Active Bridge [20]. Also, SANE was implemented in the SwitchWare environment[10] and the performance implications of providing security in active networks were studied [22]. The approach used is relatively lightweight because static checks allow the later dynamic checks to be faster or even to be eliminated. To provide a measure of the cost imposed by authentication, the costs of sending an active ping with and without authentication were compared. The ping was generated at a source machine, transmitted over a crossover cable via 100 Mb/s Ethernet to the target machine, loaded and evaluated, and then send back to the source machine, where it was again loaded and evaluated. An unauthenticated ping took an average of 5.085 ms versus 8.052 ms for the authenticated ping. Measurements of the throughput of authenticated and unauthenticated data transfer were also made. The performance degradation was 28 percent for receiving data packets and 62 percent for sending data packets.

Since there is no other work with measurements on the performance of securing active networks, there is no way to judge whether SANE performs well or not. Also, the measurements were made in a "network" of only two nodes. However, we could claim that the above results are an indication that active networks can be secure despite the increased flexibility they offer, with a reasonable performance cost.

[10] *Switchware is presented in the Architectures section.*

Security and Safety from the Programming Point of View

So far we have been concerned with security issues from a systems point of view. However, security issues from a programming point of view raise interesting possibilities. For example, a well designed programming language for active networks may solve many security problems such as eliminating run-time checks, and thus improve performance. Also, a lightweight programming language with purposely restricted functionality may even eliminate the need for the security of SANE or another relevant architecture.

The goal of a programming language for active networks is to provide security and integrity without compromising high performance. Since the programming language defines what operations the programmer can perform, by careful choice or design of a language we can limit some of the undesirable actions that a programmer might unintentionally or maliciously perform. Also, we can improve the performance because the speed of execution of the active packets depends highly on the language. Problems that may arise with their possible solutions are as follows.

- Dereferencing arbitrary areas of memory: weakly typed languages such as C are inappropriate because of the above problem. The solution to that is to use strongly typed languages such as Java.
- Allocating large amount of memory or spawning large number of sub-processes: the solution to that problem is to put bounds on the memory and the number of sub-processes that any process may have.
- Low performance due to run-time checks: implementing safety and security policies in current computing environments requires run-time checks. These run-time checks add to the overhead in executing a program, and for functions with strict performance constraints, this resulting delays may be unacceptable. Defining a language which replaces dynamic run-time checks with static checks is a solution. Therefore we can claim that a well designed typed language can remove the requirement for run-time checks.
- Low performance due to code execution: since packet forwarding should be fast, the execution of the code of the active packets should be as fast as possible. Separating the forwarding process by the other computations such as heap allocation is a way to accelerate the forwarding process.

Safetynet — Safetynet is a project on the issue of designing a programming language for active networks that has been undertaken at the University of Sussex [27, 28]. By combining recent work on semantics of computation with a pragmatic description of the processes of packet communication, researchers at Sussex have produced a programming model that both protects the connectivity of the network and places specific requirements to ensure security, safety, and fair resource allocation.

The design of the programming model is based on three major viewpoints: communication, safety and security, and programming. The model evolved enables the novel applications of active networks, while protecting the common resources of the network from both malicious attacks and buggy programs. The viewpoints place a number of requirements that are obeyed by the programming model. Some of these requirements are as follows.

Comunication requirements: Given a destination, a program should only be able to discover the next hop on the route to the destination. This is a basic characteristic of the current way routing works. Another requirement is that the next active network hop should be transparent to the actual

number of active-network-unaware nodes in between the two active network nodes. This requirement presumes that some of the nodes of the Internet will become active network aware and will form an Abone, exactly in the same manner as Mbone was formed for multicasting. A third requirement is that the state installed in a node must be reinstallable or valid when reappearing after a temporary absence. This is critical because nodes in the network are subject to arbitrary reboots and disconnections. In the context of active networks, their state should not be lost but preserved. Finally, network condition to any next hop should be available. This requirement is needed so that decisions upon possible courses of action are based on network characteristics estimates.

Safety and Security requirements:

- Bandwidth requirements: The resulting generation of packets from sending a packet into the network must be bounded. The number of packets generated per unit time from a single node must also be bounded. The above two requirements provide that a program cannot generate a denial of service attack as far as bandwidth is concerned.
- Processing time requirements: Packet forwarding code must not be able to enter an infinite loop because it should be executed as fast as possible. There must be a bound on the amount of processing time per unit time a thread[11] can consume. There must also be a bound on the number of timers per unit time a thread can set.
- Memory requirements: Since each thread takes up memory, there should be a bound on the number of threads a program can generate. Also, there should be a bound on the amount of memory a program upon a node can use. Another requirement is that programs cannot generate arbitrary references to memory. Because heap allocation takes time, it is not desirable for packet forwarding code to allocate heap. Finally, an active network program should not directly manipulate the routing table of an active node. This requirement avoids disconnections in the network but restricts the power of the active packets. However, it makes much more sense for active packets to indirectly change the routing table by calling relevant routines from the routing protocol instead of directly manipulating its contents.
- Security requirements: The security model of Safetynet is based around a set of trusted nodes, and trusted code, which is protected using cryptographic techniques. It must be possible to trace a chain of trust from a given code to a trusted node. Also, this chain must not be forgeable.

Programming requirements: The major requirement is the replacement of run-time checks by static checks. Static checks are usually more complex but they only have to be carried out once. The use of a strongly typed programming environment that embodies policies about safety and security within the type system, allows to statically prove that a program is safe. Therefore, a type system that supports the safety and security concerns of the model is also a requirement.

A language that conforms to all the above requirements may not be enough to provide security in an active network environment. However, it will remove the costly run-time checks that many current languages have. Since performance is a key issue, this is very important.

PLAN — Programming Language for active networks (PLAN) [31, 30] is an example of a language specifically deployed for active networks that tries to address the security and safety issues from a programming point of view. PLAN is a new language for programs that form the packets of a programmable network. These programs replace the packet headers used in current networks. The basic design choice of PLAN is to have programs that are lightweight and of restricted functionality. The limitations on the capabilities of PLAN are mitigated by allowing PLAN code to call other programming routines, called service routines, that reside in nodes and are written in other, more powerful languages. Since PLAN programs are lightweight, no authentication is used. However, for the service routines that reside in nodes, authentication is used when needed. PLAN addresses the issues of safety and security, performance, and flexibility as follows.

- Safety and Security: the requirements on bandwidth, processing time, and memory are addressed in two ways. First, PLAN programs are guaranteed to terminate. This is because recursive function calls and unbounded iterations are absent from the language. Second, PLAN programs have bounds on the amount of resources that they can consume. Let's visualize the maximum amount of resources consumable by a single packet on a single router as a "resource unit." Under PLAN, both the number of resource units that can be produced by a packet and the size of each resource unit are bounded by counters. More general requirements on safety and security are addressed as follows: PLAN is a purely functional and strongly-typed language. Thus, PLAN programs are statically type checked before injecting in the network, so that they don't have type errors in order to provide safety. Also, PLAN programs are pointer-safe and concurrently executing programs cannot interfere with each other. Finally, basic error handling is provided along with some service routines to be used if the former is not enough. It is interesting to note that the absence of authentication for PLAN programs is justified by their restricted functionality and the fact that if they required authentication or other costly checks before executing, they would have been prohibitively inefficient to use as packet programs.
- Performance: a major benefit of keeping PLAN simple is that its interpretation is lightweight and common tasks can be done easily and fast.
- Flexibility: PLAN is not completely general but is able to express programs for network configuration and diagnostics, and to provide the distributed computing "glue" that connects router resident service routines into larger protocols.

PLAN was used to build a non-trivial network, the PLANet. PLANet is an active internetwork in which all packets are PLAN programs. Service routines are also supported. More on the architectural considerations of this approach will be presented in the "Architectures" section.

Security is still an open issue in the area of active networking as well as to other relevant areas of research such as mobile software agents and concurrent and distributed languages with encryption and process migration features. Even if more secure testbeds that conform to the appropriate security and safety requirements are built, these testbeds should be tested in large-scale networks before rigid conclusions can be drawn regarding their performance and actual security.

ARCHITECTURES

In this section we describe some architectures for active networks. The architectures are grouped according to their basic approach toward the realization of active networking: in

[11] A thread is a part of a program that can execute independently of other parts. Operating systems that support multithreading enable programmers to design programs whose threaded parts can execute concurrently.

some architectures active packets carry executable code, usually executable on the data of the same packet that carries the code. Other architectures place the executable code in active nodes and let packets carry some identifiers to indicate which code should be executed on behalf of them. Schemes for distribution and downloading of code are developed so that not all nodes need to have all the code that they may use. Finally, some architectures give the user the opportunity to choose between lightweight code that is carried by active packets or heavyweight code that resides in the active nodes. For each architecture presented, the focus will be on its design attributes, capabilities, performance, and the way it addresses security and programmability issues.

ACTIVE PACKETS APPROACH

Most of the early active networks architectures follow the "active packets" approach, which is fundamentally characterized by the fact that the code is carried by the packets. The nodes are also active because they allow computations up to the application layer to take place, but no active code resides in them. Therefore, the reason for calling these technologies "active packets" technologies is that active code is carried by the packets either to be executed on the data of the same packet that carries the code, or to be executed in order to change the state or the behavior of the node. Examples of such architectures are the Smart Packets project proposed at BBN Technologies, the Active IP Option proposed at MIT, and the M0 architecture proposed at the University of California at Berkeley and at the University of Zurich.

Smart Packets — In the Smart Packets project [11] two important decisions were made, in an attempt to provide a rich and flexible programmable environment without overloading the computing power of the managed node and without making an environment so complex that it is difficult to be secure. The first decision is that programs must be completely self-contained, thus discounting the need for persistent states in a router. Also, programs must fit entirely into one packet — so they cannot be more than 1 Kbyte long — and the packet should not be fragmented. The second decision is that the operating environment must provide safety and security because packets containing executable code are extremely dangerous.

A special protocol called active network Encapsulation Protocol (ANEP) was developed for the DARPA active networks program to facilitate portability and interoperability among different active networks projects [34]. Management and monitoring programs generate smart packets. Smart packets are encapsulated within ANEP packets and ANEP packets are encapsulated within an IP packet. Smart packets are sent either to an end host or to each router in a hop-by-hop manner along the path to an end host. In the first case, the content of the smart packet is executed in the end host and the results are returned back and in the second case the content is executed in all the intermediate nodes. An ANEP demon located in each node is responsible for both injecting and receiving smart packets and for offering a secure environment, called virtual machine, for executing the programs.

The code of smart packets can be written in either Sprocket, a high-level language much like C, or Spanner, an assembly language. Sprocket programs are compiled into Spanner code, which in turn, is assembled into a compact machine-independent binary encoding that is placed into program packets. According to the authors, the reason why two new programming languages are used is that already existing languages could not encode more than a trivial program in the space of 1 Kbyte and that none had compact, platform-independent encodings.[12]

As far as security is concerned, smart packets achieve the correct operation of a router and its configuration by evaluating programs conservatively (i.e., if a virtual machine does not know how to handle a situation it quits execution and sends an error packet back to the source of the program), by checking whether a program comes from an authorized user, by checking the data integrity of a program in each node, and by placing limits on the execution of programs, such as offering a resource-limited environment.

Smart packets capabilities are indirectly limited by two reasons: first, the programs must be at most 1 Kbyte long; second, the functionality provided by the project is limited and tailored to network management applications. The positive part is that the performance of the technology should be good comparing to other active packets approaches. However, we are not aware of any results which prove this.

Active IP Option — The Active IP Option [35], describes an extension to the IP options mechanism that supports the embedding of program fragments in datagrams and the evaluation of these fragments as they traverse the network. The present day passive packets are replaced by active capsules, which are miniature programs that are executed as they travel. These capsules can invoke predefined primitives that interact with the local node environment, and leave information behind in a node that they have visited. Subsequent capsules can carry code that depends on this information. The capsule approach is an "in-band" approach in the sense that capsules carry the code along with the data on which it operates.

Two options are defined. The first is used to carry program fragments, which may be encoded in a variety of languages. The second is used to query an active router for the languages it supports. Backward compatibility is automatically achieved because Internet hosts silently ignore options they do not recognize. An example of a program fragment together with the format of the capsule is shown in Fig. 2.

Active options can perform routing, copying, and merging functions. The processing environment allows ambient network conditions to be examined, the current datagram to be dispatched, and additional datagrams to be constructed and sent. The state of the node may also be modified. Since the scheme is based in an extension of the IP Options mechanism, the capabilities of the technology are limited (for example, arbitrary protocols cannot be deployed). The language used in the first implementation of the architecture is TCL. The processing is done by a stripped-down TCL interpreter resulting in a restricted environment conceptually similar to that of Safe-TCL. This is the only means by which security and safety issues are addressed. Finally, the current implementation is written for functionality rather than performance.

M0 Architecture — The messenger [17] in the M0 system is similar to the capsule or the smart packet. Messengers are programs exchanged between M0 nodes. There are four elements inside the M0 node: concurrent messenger threads, a shared memory area, a simple synchronization mechanism, and channels toward neighboring nodes.

Each messenger is executed by an independent and anonymous thread of control. These threads have their own private memory space and are fully protected from each other. Mes-

[12] *Java, an obvious candidate, has programs that are much larger than smart packets can tolerate.*

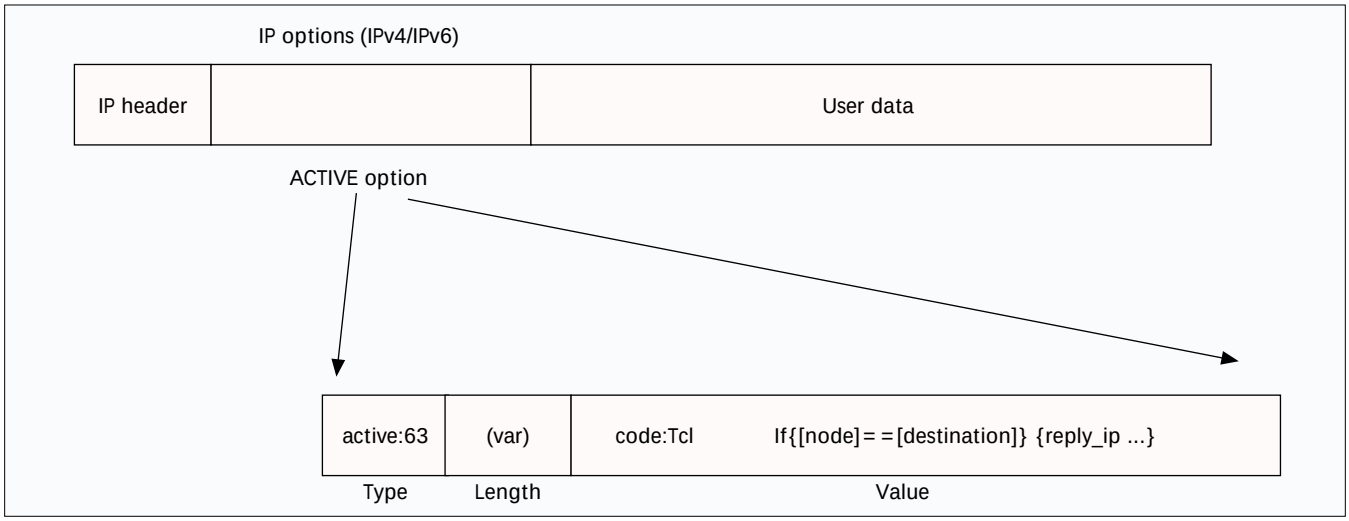

Figure 2. *Format of the capsule.*

senger threads can deposit arbitrary data structures under self-chosen names so that other threads can access them. Thread queues are a way to serialize the execution of threads in order to avoid race conditions. Channels enable messenger threads to send new messenger packets to neighboring nodes. The current M0 implementation maps messenger transmission to UDP, Ethernet, or serial-line communications.

Messenger code is written in the M0 language. M0 is a high-level language that inherits from PostScript the main concepts of operand, dictionary, and execution stack, as well as the main data manipulation and flow control operators. The M0 interpreter is written in C. M0 has no explicit code caching or code loading functionality. The code is shipped with every messenger. This works quite well for small protocols where the code is only a few bytes long. For large code sizes, messengers implement their own caching method by storing the code in the shared memory area of a node under a chosen name. This option allows the deployment of any protocol, no matter how complex it is. Therefore, the M0 architecture appears to be more powerful than the previous two architectures.

Each M0 node manages its own resources independently of other nodes. All resources have price tags which depend on the node's actual load for a given resource and also on the demand from the running threads. Messenger threads are charged for their activities. When they run out of money they are silently removed form the system. On arrival, each messenger thread obtains an account with some start money. The amount is sufficient to do some exploration inside the node and eventually send out another messenger. There is no authentication between M0 nodes, nor has a messenger any identity attached to it that would allow authentication. Safety-related questions on resource consumption have to be handled by controlling the flow of money. Access control for node-specific system resources is controlled by some agreement between a messenger and the system. M0 provides some basic cryptographic operators that can be invoked by a messenger.

ACTIVE NODES APPROACH

In the active nodes approach, the packets do not carry the actual code, but instead carry some identifiers or references to predefined functions that reside in the active nodes. The packets are active in the sense that they decide which functions are going to be executed on their data, and they provide the parameters for these functions. However, the actual code resides in the active nodes, it is not carried by the packets. This is why we call these technologies "active nodes" technologies. The motivation for such an architecture is that the active packets approach suffers from either performance-related problems because safety and security requirements are huge, or capability related problems because the only way to minimize the security and safety issues is by restricting the programs that are carried in packets (e.g., Smart Packets or PLAN packets). Examples of "active nodes" architectures are an architecture proposed at Georgia Institute of Technology, the DAN architecture proposed at Washington University and at ETH Zurich, and the ANTS architecture proposed at MIT.

An Architecture for Active Networks — In this architecture users control the invocation of predefined network-based functions through control information in packet headers [39, 40]. Users can select from an available set of functions to be computed on their data and can supply parameters as input to those computations. The available functions are chosen and implemented by the network service provider, and support-specific services. Thus, users are able to influence the computation of a selected function but cannot define arbitrary functions to be computed. This approach has some benefits with respect to incremental deployment, security, and efficiency, however, it seems to be slightly restrictive because only the functions that have been preloaded can be called upon.

Each of the functions that a node supports has a unique identifier. Each packet has a set of headers, which specify the identifier of one or more functions to be applied to the packet and parameters to be supplied to those functions. When the packet is processed, the function identified by each header is applied, resulting in updating of the node's state and possibly modification of the rest of the packet. Thus, for each function *f*, and each parameter value *p*, there is a particular subset of the node's generic state information that is relevant to *f* and parameter *p*. Functions cannot modify or use parts of the node state that are not relevant.

The strength of active networking can be realized by incremental addition of user controllable functions. Each function is precisely defined and supports a specific service. The introduction of a new active networking function involves specification of its identifier, of the parameters associated with it, and of its semantics. Once a function is specified, each provider or vendor is free to implement it in a manner consistent to the specifications.

This approach has backward compatibility in that not all users have to be aware of the active functionality in the network, and not all nodes have to support the same functions. The scheme may have low flexibility and restricted capabilities but it achieves high performance because security can be easily addressed.

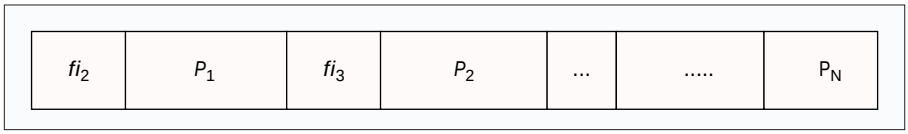

Figure 3. *Datagram (schematically).*

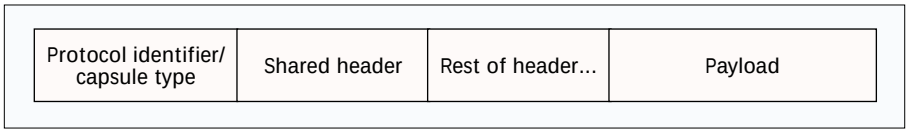

Figure 4. *Capsule format.*

DAN Architecture — The packets in Distributed Code Caching for Active Networks (DAN) architecture [44] contain a finite sequence of function identifiers, and parameters for the functions. The functions are daisy-chained in the sense that one function calls the next according to the order of the identifiers in the packet. Depending on the type of node that the packet is processed upon and the packet's content, only a subset of the functions may be called. Thus, the packet may be interpreted as a sequence of function identifiers $fi_2...fi_N$,as shown in Fig. 3, with a distinct set of parameters $P1...P_N$. The first function is not indicated by any identifier but is derived from the context in which the packet processing starts (e.g., a packet received by an Ethernet card results in the calling of Ethernet input function).

If the node is unable to locate a function, it temporarily suspends the processing of the packet and calls a "code server" for the implementation of the function. The code server is a well known node in the network which provides a library of functions for different types of operating systems from various developers. Once the module is downloaded, it is permanently stored locally on the node in order to prevent more downloads of the same module. Code servers will be put in a hierarchy for the best possible distribution of active modules. The option of downloading modules differentiates this technology from the previous one. DAN is more flexible because new functionality can be deployed and then just added to the code servers. If a node needs a new module it can easily download it. In the previous technology, the network manager should add to each node all the functions that they may need.

The active modules provided by the code servers are programmed in a high-level language such as C and compiled into object code. Once the functions are loaded by the node, they are in no way different than the ones compiled into the network at build time. Thus, all functions run at high speed and the performance is good. However, downloading a function on demand causes some delay that reduces the overall performance. A solution is to download the modules before they are needed.

Security concerns are addressed by using well known code servers which authenticate themselves and give the node the possibility to check the module's sources, and by providing digitally signed modules from well known developers only. The security problem is thus reduced to the installation of a rule which enables the node to choose the right code server, and a database of public keys to check the developer's signature. Also, even if the module sources and the modules are authenticated, network administrators may restrict the set of developers they accept modules from.

ANTS — ANTS [37] is an active network toolkit where arbitrary new protocols are automatically deployed at both intermediate nodes and end systems by using mobile code techniques. The network is viewed as a distributed programming system. The architecture introduces three components/ schemes: capsules, active nodes, and code distribution.

Capsules are replacements for a packet. Their function is to include a reference to the forwarding routine to be used to process the capsule at each active node. Therefore, references and forwarding routines in ANTS is the equivalent to identifiers and functions in DAN, respectively. Some routines are "well known" in that they are available at every active node. Other routines may be application specific. Typically, they will not reside at every node but will be transferred to a node by a code distribution mechanism before the capsules of that type can be processed for the first time. Related capsule types form a code group. The forwarding routines of a code group are transferred as a unit by the code distribution system. Related code groups form a protocol. Protocols are the units by which the network as a whole is customized by the applications. Capsules identify their type and the protocol to which they belong. When a capsule arrives at a node, a cache of protocol is checked. If the required code is not all present, a load request for the missing portion of the capsule type and protocol is sent to the upstream neighbor and the capsule is put to "sleep." When the upstream neighbor receives the request, it answers immediately (if possible) and sends the requested code. When the downstream requester receives the code, it caches it and if all the required code becomes available, it "wakes" up the sleeping capsule. If requests for code remain unanswered, sleeping capsules are discarded. On-demand loading and caching is also used in DAN. However, while in ANTS loading takes place between neighbor active nodes, in DAN loading takes place between code servers and nodes.

The format of the capsule is as shown in Fig. 4. The capsule carries an identifier for its protocol and the particular capsule type within the protocol. The identifier is based on a fingerprint of the protocol code in order to reduce the danger of protocol spoofing and also to allow protocols and capsule types to be allocated quickly and in a decentralized fashion. The remainder of the capsule has a shared header that has fields common to all capsules, a type-dependent header, and a payload. The shared header has the source and destination addresses and information about resource limits to be enforced by nodes.

The protocols need to be executed within a restricted environment that limits their access to shared resources. Active nodes play this role. During the processing, active nodes are responsible for the integrity of the network and handle any errors that may arise. Small tasks are not to be authenticated, but are to be protected by the safety mechanisms of mobile code technology, whereas use of primitives that manipulate shared logical resources, e.g., updates to the routing tables, must be authenticated. Each capsule has a resource limit that functions as a generalized Time To Live (TTL) field. This field is decreased by the nodes as resources are consumed and nodes discard capsules when their limit reaches zero. Finally, forwarding routines are expected to run to completion locally and within a short time, and their mem-

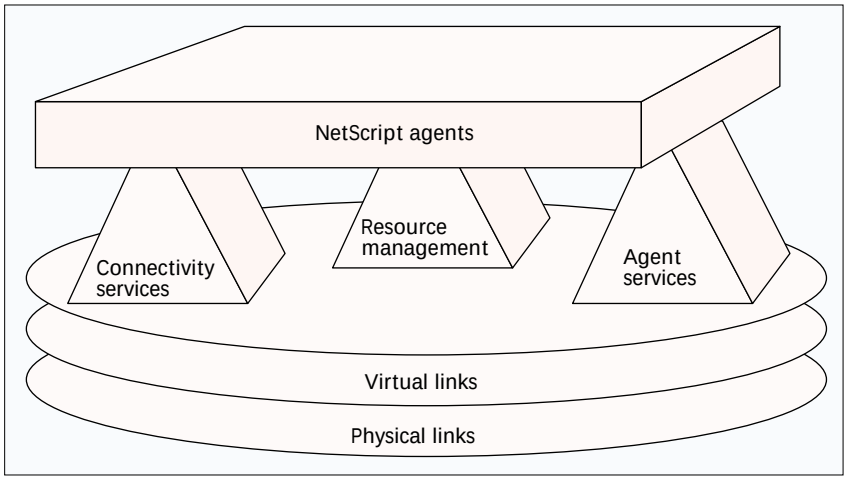

Figure 5. *Architecture of a programmable virtual network engine.*

ory and bandwidth consumption is bounded.

A Java-based prototype of ANTS has been created. The security of the implementation lies in the Java system itself. The choice of Java has allowed the researchers to evolve their architecture quickly but at the cost of less control over resources usage and lower absolute performance.

Active Packets and Nodes Approach

It should be clear by now that active packets can carry code efficiently only when the code is relatively simple and restricted. On the other hand, active nodes can efficiently provide any code. However, this code is predefined because it should reside in the active node or at least to one node from which it can be downloaded. In the active packets and nodes approach, active packets carry actual code and other more complex code resides in active nodes. Therefore, the merits of the two previous approaches exist in one system. Usually, such architectures allow users to choose either the one or the other approach according to the nature of their application. A typical example is the SwitchWare architecture proposed at the University of Pennsylvania. NetScript architecture, proposed at Columbia University, follows its own approach toward programmable networks but will be presented here as it is relatively similar to the active packets and nodes approach and fairly general.

SwitchWare Architecture — SwitchWare [36] uses a layered architecture to provide a range of different flexibility, safety and security, performance, and usability tradeoffs. The three layers defined in SwitchWare are Active Packets, Switchlets, and Active Router Infrastructure. The first layer realizes what we have called the active packets approach and the second layer realizes what we have called the active nodes approach.

In SwitchWare, active packets carry programs consisting of both code and data, and replace both the header and payload of a conventional packet. The programming language used is PLAN(Programming Language for active networks) [31]. As has been discussed,[13] PLAN is a lightweight language. It allows resource limited computation without the need for authentication, yet it performs authorized actions when required. PLAN programs are made secure by greatly restricting their actions. To compensate for this limitation, PLAN programs call routines called Switchlets, which can authenticate or use other more heavyweight mechanisms to provide security on an as needed basis. A PLAN program consists of code, plus an indication of which function should be executed first when the program arrives at a router, plus any data that makes up the arguments of that function.

Switchlets form the middle layer of the SwitchWare Architecture. The active packets were deliberately limited in power for speed, but active packets combined with switchlets can implement arbitrary protocols or functionality. Switchlets can be dynamically loaded across the network, but they execute entirely on a particular router. Thus they are base functionality or dynamic extensions rather than "mobile code." In the current implementation, switchlets are written in Caml. Switchlets can be subjected to heavier-weight security checks than active packets can. They are statically type-checked on arrival at a router and some may even carry cryptographic signatures. Switchlets can be given more latitude because of heavier checking and can access facilities in the router that active packets cannot. Thus, they can create or change the state of the router and they have direct access to the routers' network interfaces.

The active router infrastructure is the solid base upon which active packets and switchlets are build. The security of the SwitchWare architecture as a whole is granted in this layer. Below that layer is SANE, an architecture which provides a minimal set of trust assumptions, the ability to securely bootstrap the remainder of the system when the trust assumptions are met, and authentication and naming service for code that is loaded.[14]

The key point of the SwitchWare Architecture is the layered architecture with functionality partitioned between layers based on the flexibility and security tradeoffs required at each layer. Higher layers of the system provide more restricted functionality, in exchange for less security risk and very good performance. Lower layers provide arbitrary functionality but, due to the increased security issues, they are not very fast. Overall, there is a good tradeoff among security, flexibility, and performance.

NetScript Architecture — The NetScript Project [42] concentrates on the right paradigm to program networks efficiently. It provides an architecture for programming networks, an architecture of a dynamically programmable network device/node, and a language called NetScript for building network software on a programmable network. NetScript uses delegated agents to program and control the functions of intermediate network devices/nodes.

NetScript views a network as a collection of virtual network engines (VNEs) interconnected by virtual links (VLs). VNEs can be programmed by NetScript agents to process packet streams and relay these streams over VLs to other VNEs. The collection of VNEs and VLs define a NetScript virtual network (NVN). NetScript provides a language to program the NVN. A physical node may be executing many VNEs and a VL may correspond to a collection of physical links and nodes that interconnect VNEs. A VL can also interconnect any number of VNEs to handle broadcast links.

The architecture of the VNE is shown in Fig. 5. The

[13] *PLAN is described in the third section of this article.*

[14] *Refer to the section "Security" for more details.*

Agent Services layer provides a multi-threaded execution environment to support delegation, execution, and control of agent programs. It also supports message communication services among agents. The Connectivity Services module is responsible for interacting with the underlying physical environment to allocate and maintain VLs to neighboring VNEs. It provides a library of primitives used by NetScript programs to control the allocation of VL resources, and the scheduling and transmission of packets over VLs. Packets contain a minimal NetScript encapsulation header that identifies the stream to which they belong. When a packet arrives at a VNE, this header is used by the run-time environment to pass it to the respective programs which process this stream. The active packets of the scheme are the NetScript packets and the active nodes are the VNEs.Communication services provided by the VNE are entirely local and permit interaction with neighboring VNEs only.

NetScript language is a dynamic dataflow language designed specifically for communications-based tasks. It can operate on streams of packets. It is based on simple object-oriented principles, so that programmers can override default operators with customized versions of their own. A NetScript program consists of a pool of communicating threads. These threads communicate through message streams that connect inputs to outputs of executing programs. Communicating programs can be geographically distributed. NetScript provides a universal abstraction of a programmable networking device because constructs hide the heterogeneity of networking devices behind simple abstractions.

The main difference between NetScript and other architectures is the focus on the programmability of networks. Here, the main assumption is that a single language based on the right model can greatly simplify protocol construction and allow flexibility in experimenting with appropriate programming features. Another difference is that NetScript treats the network as a single programmable abstraction rather than an heterogeneous collection of programmable intermediate nodes and end-nodes.

COMPARISON

The active packets approach suffers from performance-related problems because safety and security requirements are huge. In an effort to reduce the security burden and thus increase performance, some researchers have decided to restrict the functionality of the programs carried by the active packets, resulting in architectures with decreased capabilities. M0 is the only architecture within the active packet approach that can provide arbitrary functionality, thanks to its novel caching technique.

The active nodes approach has good performance because security issues are much less than in the previous approach. However, the flexibility of the relevant architectures is limited. In an effort to increase flexibility, DAN and ANTS architectures have adopted a scheme where code is downloaded on demand and is cached for future use. As a result, these two technologies can easily deploy any new arbitrary protocol. Nevertheless, downloading code on demand causes some delay that reduces the overall performance.

The combination of both approaches seems to be very appealing. SwitchWare architecture realizes this idea by the use of a layered architecture, and manages to provide a range of different flexibility, safety and security, performance, and usability tradeoffs. Finally, NetScript architecture proposes a novel viewpoint where the network is treated as a single programmable abstraction.

CONCLUSIONS

We have indicated various applications where active networks can be beneficial. Network management, congestion control, reliable and efficient multicasting, and active caching are some of them. Current research in applying active networks in the above application domains, as well as proposed architectures that support them, assist the examination of active networks usefulness, applicability, and efficiency. In this article we presented: Network Management by Delegation, Darwin Project (an experimental technology related to congestion control), ARM, Self-Organizing Wide-Area Network Caches, Secure active network Environment, SafetyNet, PLAN, Smart Packets, Active IP Option, M0 architecture, Distributed Code Caching for active networks, ANTS, SwitchWare Architecture, and NetScript Architecture. Finally, we have addressed the problem of security of such a flexible infrastructure as well as programmability issues that stem from the need for security without compromising high performance.

Active networking is undoubtedly an exciting step in network design. It has the potential for solving many of the problems identified in current "passive" networks, and has a wide application range. However, the implementations that have been built at the various research sites so far have not been tested in large-scale networks. It is not directly clear how the limited data that are available would generalize to a large-scale network like the Internet. Before attempting any cogent conclusions, it would be necessary to deploy an Abone, analogous to the Mbone deployed for multicasting, and test active network technologies in real conditions.[15] Last but not least, two tradeoffs exist in active networking: a tradeoff between security levels and performance, and a tradeoff between usability/flexibility and complexity. A lot more can be done to address the security and programmability issues and it is very difficult at the moment to draw the line where the security-performance tradeoff can be optimized. "Conservative" approaches toward active networks may yield satisfactory results, particularly in combination with the mobile software agent technology that is currently emerging. The usability/flexibility of such active networks would be somehow compromised, but complexity would be manageable. The optimal line of the flexibility–complexity tradeoff is also hard to draw. Abone may be used to find optimum solutions for both tradeoffs.

Irrespective of anyone's thoughts and concerns, research on active networking is already on its way. Its success would mean that intermediate node functions may be programmed and deployed through simple, open, and rapid processes requiring no standards committees' or vendors' resources. Success will furthermore mean the possibility for automatic upgrade of network protocols. Given these two features, it is well worth trying. In any case, perhaps the problem is not whether networks should be programmable or not, but deciding on paradigms that will program them efficiently.

ACKNOWLEDGEMENTS

I would like to express my thanks to the anonymous reviewers for their insightful and helpful comments. I would also like to thank Eleutheria Psouni for helping me in proof-

[15] Recently, several active network technologies have been deployed at different cites to form the DARPA-sponsored Abone, an experimental active network in which nodes communicate by tunneling through the Internet using UDP.

reading the final draft. Finally, I wish to acknowledge Labhesh Patel for his input and discussions regarding active network architectures.

REFERENCES

[1] David L. Tennenhouse and David J. Wetherall, Towards an Active Network Architecture, in Multimedia Computing and Networking (MMCN '96), 1996, San Jose, CA.
[2] Ulana Legedza, David Wetherall, and John Guttag, Improving the Performance of Distributed Applications Using Active Networks, *Proc. IEEE INFOCOM '98,* San Francisco, CA, 29 March–2 April 1998.
[3] D. L. Tennenhouse *et al*, From Internet to ActiveNet, request for comments, Jan. 1996.
[4] D. L. Tennenhouse *et al.*, A Survey of Active Network Research, *IEEE Commun. Mag.*, vol. 35, no. 1, 1997.
[5] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura, Active Networking and the End-to-End Argument, *Proc. ICNP '97,* Atlanta, GA, Oct. 1997.
[6] Samrat Bhattacharjee *et al.*, Commentaries on "Active Networking and End-to-End Arguments," *IEEE Network*, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3.
[7] Ahmed Karmouch, Mobile Software Agents for Telecommunications. Guest Editorial for *IEEE Commun. Mag.*, July 1998 vol. 36 no. 7.
[8] Vu Anh Pham and Ahmed Karmouch, Mobile Software Agents: An Overview, *IEEE Commun. Mag.*, July 1998, vol. 36, no. 7.
[9] Michael S. Greenberg, Jennifer C. Byington, and David G. Harper, Mobile Agents and Security. *IEEE Commun. Mag.*, July 1998, vol. 36, no. 7.
[10] Prashant Chandra *et al*., Darwin: Customizable Resource Management for Value-Added Network Services, *Proc. Sixth IEEE Int'l Conf. on Network Protocols (ICNP '98),* Austin, Oct. 1998.
[11] Beverly Schwartz, Wenyi Zhou, and Alden W. Jackson, Smart Packets for Active Networks, BBN Technologies, Jan. 1998.
[12] Prashant Chandra *et al*., Network Support for ApplicationOriented QoS, Sixth IEEE/IFIP International Workshop on Quality of Service, Napa, May 98.
[13] Y. Yemini, G. Goldszmidt, and S. Yemini, Network Management by Delegation, Integrated Network Management II, I. Krishnan and W. Zimmer, Eds., pp. 95–107, North-Holland, 1991.
[14] G. Goldszmidt and Y. Yemini, Delegeted Agents for Network Management, *IEEE Commun. Mag.*, March 1998, vol. 36, no. 3, pp. 66–70.
[15] A. Vassila, G. Pavlou, and G. Knight, Active Objects in TMN, *Integrated Network Management V*, A. Lazar, R. Saracco, R. Stadler, Eds., pp. 139–150, Chapman & Hall, 1997.
[16] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura, On Active Networking and Congestion. Technical Report GIT-CC-96/02.
[17] Albert Banchs *et al*., Multicasting Multimedia Streams with Active Networks, ICSI technical report 97-050.
[18] Li-wei H. Lehman, Stephen J. Garland, and D.L. Tennenhouse, Active Reliable Multicast, *Proc. IEEE INFOCOM '98,* San Francisco, CA, 29 March–2 April 1998.
[19] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura, Self-Organizing Wide-Area Network Caches, *Proc. IEEE INFOCOM '98*, San Francisco, CA, 29 March–2 April 1998.
[20] D. Scott Alexander *et al*., Active Bridging, *Proc. ACM SIGCOMM '97*, Cannes, France.
[21] Christian F. Tschudin, Active Network Overlay Network (ANON), RFC Draft, December 1997.
[22] D. Scott *et al*., Performance Implications of Securing Active Networks, Technical Report MS-CIS-98-02.
[23] William A. Arbaugh *et al*., Automated Recovery in a Secure Bootstrap Process, Network and Distributed System Security Symposium, Internet Society, March 1998.
[24] Garl A. Gunter and Trevor Jim, Design of an Application-Level Security Infrastructure, DIMACS Workshop on Design and Formal Verification of Security Protocols, Sept. 3–5, 1997.
[25] D. Scott *et al*., A Secure Active Network Environment Architecture, *IEEE Network*, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3.
[26] Tomas Sander and Christian F. Tschudin, Towards Mobile Cryptography, ICSI technical report 97-049.
[27] Allan Jeffrey and Ian Wakeman, A Survey on Semantic Techniques for Active Networks, http://www.cogs.susx.ac.uk/projects/safetynet
[28] Ian Wakeman *et al*., Designing a Programming Language for Active Networks, http://www.cogs.susx.ac.uk/projects/safetynet, submitted to Hipparch special issue of *Network and ISDN Systems*, Jan. 1999.
[29] George C. Necula, Proof-Carrying Code, *Proc. 24th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, ACM Press, 1997.
[30] M.Hicks *et al*., PLANet: An Active Network Testbed, http://www.cis.upenn.edu/switchware/papers/planet.ps
[31] M.Hicks *et al*., PLAN: A Programming Language for Active Networks, http://www.cis.upenn.edu/ switchware/papers/plan.ps, submitted to ICFP '98.
[32] John Hartman *et al*., Liquid Software: A new Paradigm for Networked Systems, Technical Report TR96-11, Department of Computer Science, University of Arizona, 1996.
[33] David Wetherall, Chris Lindblad, and Henry Hough, Active Pages: Intelligent Nodes on the World Wide Web, *Proc. 1994 World Wide Web Conf.*, Geneva, Switzerland, May 1994.
[34] D. Scott Alexander *et al*., Active Network Encapsulation Protocol (ANEP), RFC Draft, July 1997.
[35] David J. Wetherall and David L. Tennenhouse, The ACTIVE_IP option, In the 7th ACM SIGOPS European Workshop.
[36] C. A. Gunter, S. M. Nettles, and J. M. Smith, The SwitchWare Active Network Architecture, *IEEE Network*, special issue on Active and Programmable Networks, May/June 1998, vol. 12, no. 3.
[37] D. J.Wetherall, J. V. Guttag, and D. L.Tennenhouse, ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols, *Proc. IEEE OPENARCH '98*, April 1998.
[38] A. B. Kulkarni *et al*., An Active Network Architecture for ATM WANs, presented at the Mobile Multimedia Commun. Conference at Princeton, NJ, Sept. 25–27, 1996.
[39] S. Bhattacharjee, K. L.Calvert, and E. W.Zequra, Implementation of an Active Networking Architecture, Technical Report, Georgia Institute of Technology, July 1996.
[40] S. Bhattacharjee, K. L.Calvert, and E W.Zequra, An Architecture for Active Networking, *Proc. IEEE INFOCOM '97*, April 1997.
[41] D. S. Alexander, A Generalized Computing Model of Active Networks, thesis report, University of Pennsylvania.
[42] Y. Yemini and Sushil da Silva, Towards Programmable Networks, *Proc. IFIP/IEEE Int'l Workshop on Distributed Systems, Operations, and Management,* L'Aquila, Italy, 1996.
[43] C. Partridge, T. Mendez, and W. Milliken, Host Anycasting Service, RFC 1546, Nov. 1993.
[44] Dan Decasper and Bernhard Plattner, DAN: Distributed Code Cashing for Active Networks, *Proc. IEEE INFOCOM '98*, San Francisco, CA, 29 March–2 April 1998.

BIOGRAPHY

KONSTANTINOS PSOUNIS (kpsounis@leland.stanford.edu) was graduated from the Electrical Engineering and Computer Science Department of National Technical University of Athens, Greece, in June 1997. He received an M.S. in electrical engineering from Stanford University, California, in December 1998, as a Stanford Graduate Fellow. His research interests are in the area of computer networks.