Active Networking: One View of the Past, Present, and Future

Jonathan M. Smith, *Fellow, IEEE,* and Scott M. Nettles, *Member, IEEE*

Abstract—**All distributed computing systems face the architectural question of the location (and nature) of programmability in the telecommunications networks, computers, and other peripheral devices comprising them. The perspective of this paper is that network elements should be as programmable as possible, to enable the most flexible distributed computing systems.**

There has been a persistent confluence among operating systems, programming languages, networking and distributed systems. We demonstrate how these interactions led to what is called "active networking," and in the spirit of "*vox audita perit, littera scripta manet*" (*the spoken word perishes, but the written word remains*), include an account of how it was made to happen. Lessons are drawn both from the broader research agenda, and the specific goals pursued in the SwitchWare project. We speculate on likely futures for active networking.

Index Terms—**Communication system software, communication systems, programming operating systems, protocols.**

I. Introduction

The BASIC goals of active networking (AN) are to create networking technologies that, in contrast to current networks, are easy to evolve and which allow application specific customization. To achieve these goals, AN uses a simple idea, that the network would be easier to change and customize if it were programmable. Although there are a few pioneering efforts [1]–[3] that predate it, AN became a vigorous research area when DARPA began funding research in the mid-1990s.

This paper is an attempt to gain some insight into AN by looking back at its origins and looking forward toward its future. The paper's form is a history of AN as seen from the authors' point-of-view as part of the SwitchWare project. The paper is idiosyncratic in style, offering in places history which might not otherwise be recorded to illustrate the way in which a research program evolves by fits and starts into a larger agenda. Additional personal recollections can be found elsewhere [4]. We have focused on the issues we are best able to comment on and omission of some event or piece of work says nothing about its importance, only that it was not our focus.

Manuscript received August 29 2002; revised March 31, 2003. This work was supported by DARPA under Contracts N66001-96-C-852 and DABT63-95-C-0073, and the National Science Foundation under CAREER Grants CCR-9702107, ANI-9906855, ANI98-13875, ANI00-82386, and ANI-0081360. This paper was recommended by Guest Editors W. Pedrycz and A. Vasilakos.

J. M. Smith is with the CIS Department, University of Pennsylvania, Philadelphia, PA 19104-6314 USA (e-mail: jms@cis.upenn.edu).

S. M. Nettles is with The University of Texas at Austin, Austin, TX 78712 USA.

Digital Object Identifier 10.1109/TSMCC.2003.818493

A. Operating Systems, Programming Languages, Distributed Systems, and Networks

One perspective we wish to develop is a view of AN as the confluence of ideas from operating systems, programming languages, networks, and distributed systems. At a high level, this is shown by the timeline in Fig. 1. At this point in the paper, this figure is mostly suggestive, but as we proceed, we will explain its pieces.

The reason that AN is a combination of ideas from each of these areas is clear. First, the basic goal of AN is to build networking systems, as such networking is the core discipline that is built upon. Further, AN focuses on how to build networks. But we can view networks as low-level distributed systems. Thus, building networks is building distributed systems, and thus, issues and ideas from distributed systems come into play. Programming languages research becomes important because we wish to build these low-level distributed systems so that they can be programmed and programming languages are the key to expressing programs. PL plays a role not just in what we can express, but also in what can not be expressed, thus giving us control over the power of programmability. Finally, two critical issues, security and resource allocation and control, motivate the operating systems (OS) role, as the focus of OS has been the abstraction, protection, and management of system resources.

B. Further AN Concerns

Although AN has the high-level goals of improving evolvability and customizabilty, there are a number of low-level concerns that must be balanced to achieve these high-level goals. The first of these concerns is flexibility. AN systems aim to significantly improve the flexibility with which we can build networks. The second concern is safety and security. It is crucial that while adding flexibility, we not compromise the safety or security of the resulting system. The third concern is performance. If adding flexibility results in a system that can not achieve its performance goals it will be pointless. The final concern is usability. It is important that the resulting system not be so complex as to be unusable. The other main perspective we wish to develop is how the combination of disciplines discussed above come together to help address these concerns.

C. Outline of the Paper

The paper begins in Section II by looking at the technologies and challenges faced by distributed computing systems of the 1970s and 1980s. This frames much of the technological evolution (and research) which has gotten us to where we are today, in

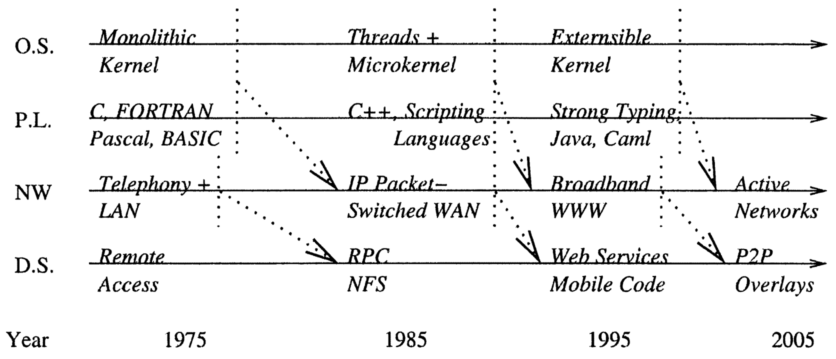

Fig. 1. Cross-coupled influences: operating systems, programming languages, distributed systems, and networking.

particular the improvements in distributed computing that could follow from a more flexible network architecture. These concerns about network architecture, discussed in Section III, led to the "store-translate-and-forward" (STF) model of networking. It is the STF model which led to the perspective of network architecture as being *driven by distributed computing* rather than being engineered in isolation. Two examples of this perspective are the design of protocol boosters (Section IV) which stimulated the DARPA effort in AN (Section V), and our contribution to that effort, the SwitchWare project (Section VI). Section VII gives our view of future work and Section VIII concludes the paper.

II. Early Code Movement Influences

One of the basic techniques of AN is that code is moved to the node at which it should execute. One place this idea first arose was in distributed systems supporting process migration. Another significant early influence was in generalization of remote procedure call (RPC) to support remote evaluation (RE).

A. Process Migration

The earliest implementation of a process migration system was in David Farber's DCS system [5], [6] in the mid-1970s. Over the course of the 1980s, a large number of researchers attempted to realize process migration with varying degrees of success (Smith [7] provides a survey). The basic motivation was that in a distributed system, it was reasonable to allocate processes to processors based on local resources such as capacity or locally-stored data.

By the late 1980s, there was considerable progress on the design and implementation of these systems in the context of operating systems, but little application support and no real support for heterogeneity. For example, in 1986, Smith implemented a system in which a process could migrate itself by checkpointing, transporting, and uncheckpointing. In the initial version of the system the executable image was shipped directly to a server, but in an enhanced version [8], NFS was used to save the image while sending the image's name with a single user datagram protocol (UDP) datagram to a server. The server then used the name to fetch the code and continue execution (the later is analogous in some ways to the approach pursued in active network transport system (ANTS) [9]).

The system suffered from some fairly severe constraints, e.g., that it needed a daemon, could not migrate active I/O such as pipes or sockets, had no support for heterogeneity, etc. Its major technical contribution was its demonstration that user-level process migration was possible, providing a new avenue toward writing distributed applications.

B. Remote Evaluation

The introduction of RPC [10], [11] was the first major attempt to combine ideas from programming languages and distributed systems. RPC made distributed systems programming easier because it allows remote functionality to be dealt with using a familiar and convenient interface.

By the mid-1980s some of the most sophisticated uses of RPC systems were to support distributed window systems. A particularly notable system was the network extensible window system (NeWS) [12] built by J. Gosling at Sun Microsystems. NeWS took its graphics model and programming language from Postscript. In Postscript, code to print a page was downloaded into a printer and then executed, producing a printed page as a side effect. In NeWS, Postscript for the user interface could be downloaded into the graphics server. Besides giving great flexibility, this allowed significant optimizations when the downloaded code could be used to eliminate round-trips between the (remote) client and server. For example, grabbing a group of lines and rubber-banding them could be done without going out to the network.

In the late 1980s, researchers interested in distributed programming languages began to explore the idea of generalizing RPC along the directions seen in NeWS [13]–[16]. The resulting remote evaluation systems had a simple model of computation, code could be shipped to a remote node and then evaluated. RPC is a special case where the code that is shipped is just a function call and its arguments. Both Stamos [13]–[15] and Clamen [16] used Lisp dialects as the basis for their remote evaluation systems because Lisp made representing programs (and most data) in a format that could be sent from machine to machine (ASCII byte streams) simple. Such representations also had the advantage of support for heterogeneous machine types [17], a problem for most process migration schemes.

In a very deep sense, mobile code schemes such as remote evaluation models provide a general solution to "process migration" once appropriate language technologies became available. Active networking is simply an application of these mobile code techniques and technologies to the domain of networking.

III. Broadband Internet

In the early 1990s [18], the Internet was coming into its own. A collection of researchers were exploring methods for increasing network throughputs by a factor of 100, using a variety of technologies, such as synchronous optical network (SONET) [19], [20], high-performance parallel interface (HIPPI) [21]–[23] and asynchronous transfer mode (ATM) [24], [25]. For example, research in the AURORA gigabit testbed [26], [27] was centered around ATM technology. While ATM signaling never quite matured, ATM link layers led to the broadband Internet [28], both by providing an infrastructure for high-speed Internet protocol (IP) overlays, and then later evolving into the methodology for building high performance IP switches [29]. Once ATM signaling was replaced in architectures such as multiprotocol label switching (MPLS) [30], which provided virtual paths without heavyweight signaling, the basic advantages of the technology became available to Internet users.

Among the possibilities for a broadband Internet were those of building wide area distributed computing infrastructures. Low bandwidth in the core had inhibited access to remote data, and the ability to migrate processing within the network had really not been achieved. The notion was that the availability of very high performance networking would allow large scale distributed computations, such as distributed chemical analysis and weather modeling, that were unachievable without access to remote computational resources and data.

A. Interoperability

The Internet [31] provides a universal networking infrastructure [32] by providing an interoperability layer, the IP packet format, which all network participants must use and accept. This makes the problem of sending data from an arbitrary device to an arbitrary device via an arbitrary network manageable. The sender formats an IP packet and encapsulates it in one or more frames of an attached network type. Intermediate IP-compliant devices extract the packet from an incoming frame, interpret the IP packet, and then again encapsulate the packet in a frame targeted at the ultimate destination.

While ATM made an attractive subnetwork technology, it did not solve the interoperability problem, and IP had an implemented signaling infrastructure. ATM systems provided fine-grained multiplexing in support of multimedia, and provided one solution to the link performance problem, but inadequately addressed the control plane represented by signaling protocols.[1]

Sincoskie [37] observed that the telephone network achieved interoperability with a circuit model based on a 20–mA copper loop, and the IP network achieved interoperability with this common packet format model. Each were reaching limits in terms of the cost and ability to introduce new services *into* the network.[2]

[1]This later stimulated an energetic line of research [33]—[36] at the University of Cambridge, which developed virtual signaling stacks to allow customization. The Cambridge work, "open signaling," exploited the capability of the virtual circuit identifier (VCI) in an ATM network to be used for early demultiplexing. Groups of VCIs can be associated with particular signaling software implementations, essentially creating virtual switches, one per group of addresses. Since the focus of flexibility and extensibility is the out-of-band signaling software, open signaling can be viewed as an active networking approach which is limited to the network control plane.

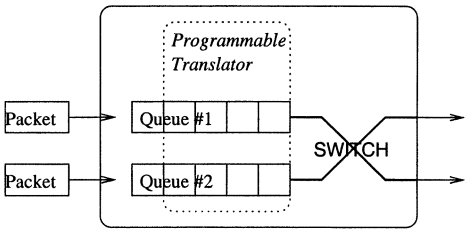

Fig. 2. STF network model.

B. Store-Translate-and-Forward

How could one extract the best features of each solution, apply the lessons learned, and apply them to a new architecture in which service introduction could be accelerated?

This problem was originally framed as "interservice networking," with goals including tying together services such as voice, fax, and IP. While the performance challenges of broadband networking were interesting, this larger architectural question became increasingly intriguing.

Smith [4], [38] addressed this problem with a generalization of store-and-forward packet-switching called store-translate-and-forward (STF), where the effect of a translator "edits" the packet data as it passes through the STF node, as is shown in Fig. 2.

The STF networking model [38] provided a strong "computer science" flavor to networking. Elementary computability relates translation (or language recognition) to computing, and activities such as route lookup, label-switching and packet-filtering were easy to represent in this model.

An experimental approach to investigate STF was not immediately obvious, but it was clearly possible simply by inserting a programmable general purpose computer into the network—a Turing machine can support a large variety of interesting translations! Inserting translators into the network was therefore at least possible, and there appeared to be interesting applications.

The question then became one of how to proceed, how to make the vision happen, and how to enable new network services.

IV. Protocol Boosters

The first outgrowth of the STF ideas was a DARPA proposal on the topic of "Protocol Boosters for Distributed Computing Systems." The idea was to *dynamically construct* protocols using protocol element insertion and deletion on an as-needed basis, to respond to network dynamics. Protocols are constructed *optimistically*; that is, ideal operating conditions (no errors, low delays, adequate throughput, etc.) were assumed, and protocol elements (such as error detection and correction mechanisms) were inserted into protocols on-demand, as conditions were encountered that deviated from the best case where the protocol element would not be needed. Such "boosters"

[2]The most successful approaches to introducing new services which later emerged, such as the World Wide Web and peer-to-peer, avoided this issue by operating as overlays.

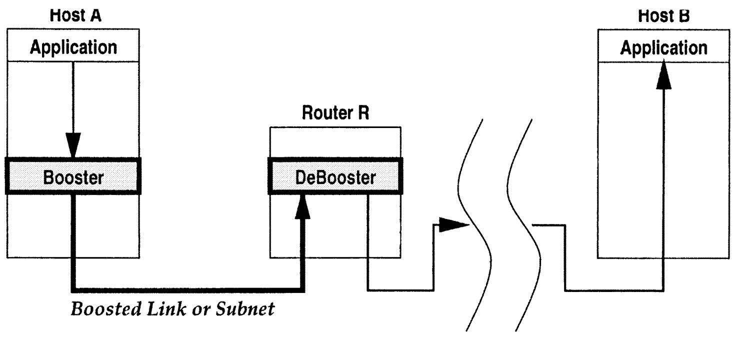

Fig. 3. Use of a protocol booster on a network link.

were a limited form of network translator. Ultimately, protocol boosters support for on-the-fly protocol deployment was adopted as a key aspect of AN.

This proposal was a significant break from convention in that it dynamically altered the network stack *in response to dynamic network conditions*. A major goal was the ability to accelerate network evolution (e.g., with proprietary protocols; one prediction was creation of a "marketplace" of protocols, where users would select and deploy their own protocols, including network-layer protocols. One important consequence of users deploying their own network-layer protocols was that significant attention had to be paid to issues of safety and security, since the network fabric was shared. A marketplace that was more likely to develop was one of proprietary protocols developed by *protocol vendors*, to be sold, or to be deployed in their own networks to achieve a competitive advantage. However, the Protocol Boosters research effort focused mainly on whether protocols could be constructed using "as-needed" techniques, and if so, what form these protocols would take.

A. Boosted Protocols

The canonical example of protocol boosters is forward error correction (FEC). FEC is a technique that uses extra information in a message to allow recovery of the message if a portion of the message is lost or damaged. In the case of a data packet, such information might include additional packets or extra information within the packet itself which might be used for recovery. Fig. 3 illustrates how the performance of a link might be boosted by protocol elements augmenting the existing protocol architecture.

FEC provides an attractive example for "as-needed" functionality, since it is in essence pessimistic. The extra bits are sent under the assumption that things will go wrong, and the efficient encoding and recovery of the message consumes processing cycles as well. Thus, one would like to insert FEC when it was providing some benefit, but not otherwise use it. Of course, when FEC will be needed is unpredictable, so a mechanism that deploys it "on-the-fly" is needed, making Protocol Boosters an ideal candidate.

B. Design Influences

The two most important influences on the initial implementation of protocol boosters were Hutchinson and Peterson's [39] x-Kernel system and Dennis Ritchie's STREAMS [40] architecture. Ritchie's system provided an elegant architecture

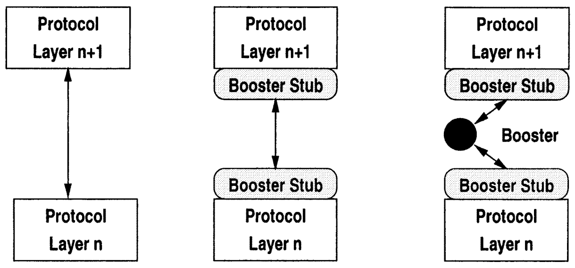

Fig. 4. Inserting a booster into a protocol stack.

for constructing protocols, later delivered with, and used heavily in, the AT&T System V version of UNIX. The initial notion had been one of stackable "line disciplines" for UNIX, but was generalized into stackable protocol architectures for streams of data. Stackable meant that code adhering to a message-handling discipline shared by all such STREAMS modules could be pushed onto, and subsequently popped from, a logical stack of processing modules through which streams of message data would pass. These modules could be dynamically inserted and removed while the protocol was in operation. This definitely had the right flavor for what was envisioned for protocol boosters, but we felt the programming model was (no doubt on purpose) too restrictive. The x-Kernel, on the other hand, was almost completely flexible, and arbitrary protocols could be constructed using the system as a basis. The x-Kernel composed a *protocol graph* of protocol components together into a system. This style of protocol composition was more in keeping with the Protocol Booster ideas, but the existing x-Kernel tools did not permit dynamic reconfiguration of the protocol graph. Similar ideas were explored by Tschudin [41] and Plagemann, *et al.* [42].

C. Infrastructures and Experimental Results

The first booster implementation [43] used a modified version of the FreeBSD operating system. Boosters were injected into and removed from the IP stack, accomplishing among other things compression, encryption and keyword filtering. The basic modification of the 4.4 BS-DLite stack is shown in Fig. 4; further modifications (basically a small multithreading system) were made to handle more complex boosters, as shown in Fig. 5.

The initial prototype showed that a flexible and dynamically modifiable system could be built, and led to a far more mature

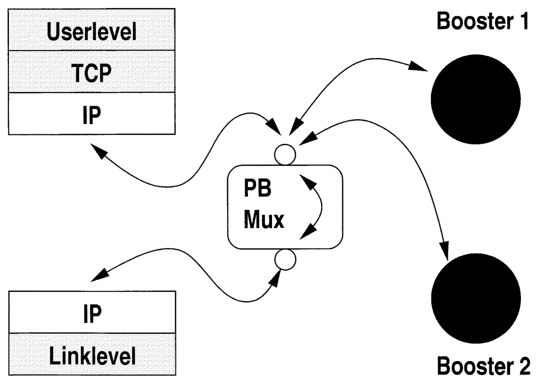

Fig. 5. Multiplexing of boosters in the 4.4 BSDlite IP stack.

kernel infrastructure [44], [45] built in collaboration with Bellcore. The Bellcore team developed a number of useful applications of the technique in the error detection and correction domain; the Bellcore implementation was used in satellite and wireless network trials.

D. High-Performance Boosters

Given that protocol boosters were used for many bit-intensive tasks (FEC, compression and encryption) their implementation in software presented performance challenges. This led to taking the basic boosters ideas and building hardware support [46], in the form of a switched pipeline of field programmable logic called the "Programmable Protocol Processing Pipeline" or P4. A photograph of the P4 is shown in Fig. 6.

The P4 was used standalone to demonstrate that boosters could be run at OC-3c line rates of 155 Mb/s, and also in concert with a PC used as a control element for inserting and removing boosters from the protocol processing path. The hybrid hardware/software architecture of Hadzic's [47] dissertation demonstrated that an FEC booster could automatically detect the need for insertion using failed AAL-5 cyclic redundancy check (CRCs), insert itself, and enhance the performance of a set of TCP/IP/ATM benchmarks, thus demonstrating the power of the protocol architecture.

V. Active Networks

In this section, we show how the ideas of the previous section (Section IV) influenced research directions, and discuss the programmatic history of AN.

A. From Protocol Boosters to DARPA's AN Program

The need for a general purpose infrastructure with which new protocols could be developed was becoming clear, something that was not a main focus of protocol boosters. The flavor of STF in protocol boosters, and its implications for evolving network infrastructures, had attracted the attention of DARPA managers thinking about network/computer integration (NCI). Protocol boosters offered a concrete proposal to achieve this. Protocol boosters provided a concrete demonstration to DARPA that approaches to NCI were possible.

DARPA charged an Information Science and Technology (ISAT) study group with defining a research agenda for NCI, which led to the DARPA AN program.

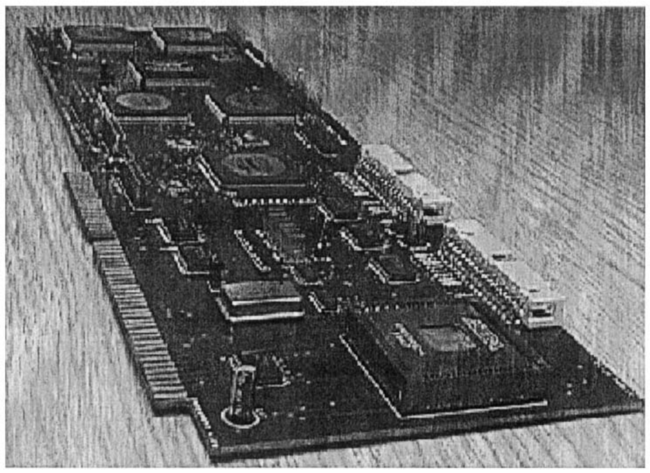

Fig. 6. Programmable protocol processing pipeline.

B. Putting Together a Program

Simultaneously with the early Protocol Boosters work, David Tennenhouse had begun to think about plans for a programmable network infrastructure, which he called "Active Networks." He had also had been serving on the ISAT team that was considering ideas involving programmable networks. Gary Minden at DARPA was successful in using the results of the ISAT study to create a "broad agency announcement" (BAA), DARPA's means for soliciting research proposals. This BAA, BAA 96-06, "research topics in high performance distributed services," was the watershed event that lead to the explosion of AN work in the late 1990s.

One group of researchers [48]–[53] generated a collaborative set of responses to the BAA. Teleconferencing in the Fall of 1995 was used develop a "matrix" of contributions each laboratory could make to the overall goal of a programmable network infrastructure. The matrix represented each lab's contributions as rows with varying degrees of contribution to each of the following six areas:

1) enabling technologies,
2) platform development,
3) programming models,
4) middleware services and applications;
5) active controls and algorithms;
6) network operations.

It was clear that all of these were needed, and so the goal was to ensure that no necessary research was left out of the program. Several groups agreed to include a representation of the matrix in their proposals so that DARPA could see how participants could fit together into an overall research agenda. This research agenda was later captured in a paper by Tennenhouse and Wetherall [52].

We should mention that the work that grew out of the DARPA BAA was predated by several pieces of early work that clearly foreshadow AN. First, there is the work done by Zander and Forchheimer of the University of Linköping, Sweden, on a system called "Softnet." The Softnet [2], [3] system was a packet radio network where packets of multithreaded M-FORTH code were interpreted by network elements consisting of two-processor nodes; one serviced network events, and the other ran user programs. The nodes were supported by a small operating system, which protected the network elements, e.g., to prevent buggy programs from destroying the packet-switching fabric. The focus was proof-of-concept rather than a wholesale change in network infrastructure, models and run-time support. Nonetheless, this team should be given due credit as the progenitors of what we now call AN.

Second, at around the same time, the progenitor of the "capsules" [54] or "active packet" model [55] was being developed by David Wall [1]. In his paper, Wall outlined a new approach to networking. Quoting from the paper's abstract:

"Network algorithms are usually stated from the viewpoint of the network nodes, but they can often be stated more clearly from the viewpoint of an active message, a process that intentionally moves from node to node."

While neither the softnet nor the active message systems captured the entire AN agenda, they had the basic foundations. The advances then, would be from new technologies that had arisen since, in particular new programming language [56], [57] and security technologies [58] that could provide desirable sets of tradeoffs amongst security, programmability, usability, and performance.

VI. SwitchWare

Having examined some of the trends that led to the wide scale development of AN, we now consider one specific project and its exploitation of the interplay of operating systems, distributed systems, programming languages, and networking. We place a particular emphasis on how it addressed the tension between flexibility, safety and security, performance, and usability.

We focus on the SwitchWare project [59], [60] for a number of reasons. The most obvious is that it is the work with which the authors are most familiar. However, there are other significant reasons to examine SwitchWare as well. First, SwitchWare has touched upon many of the key issues, ranging from the development and use of modern programming language technology to the development of new operating systems technologies focused on resource control. In fact, a specific initial goal of SwitchWare was exactly the application of modern programming language (PL) techniques to AN. Second, because SwitchWare is really many smaller projects, a significant number of different flexibility/security/performance tradeoffs have been explored. Finally, SwitchWare is the only project to explore (and in fact, fuse) the two main AN approaches of programming network nodes using downloaded dynamically linked "active extensions" and programming nodes by executing "active packets" that carry their code as part of the packet.

Early in the SwitchWare work, it became clear that we were interested in exploring both the active extension and active packet approaches. As our work progressed, however, we came to the conclusion that the two approaches are synergistic. This led to the SwitchWare architecture [61], an abstract model of which is presented in Fig. 7, upon which we will elaborate in this section of the paper.

In this architecture, nodes are both extensible using Active Extensions and programmable using active packets. Active

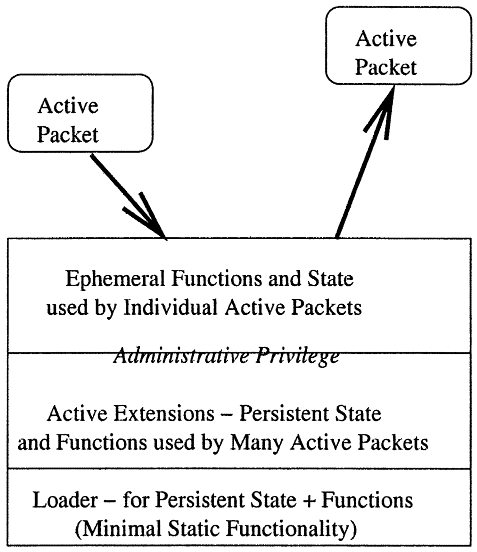
Active Packet
Active Packet
Ephemeral Functions and State used by Individual Active Packets
Administrative Privilege
Active Extensions – Persistent State and Functions used by Many Active Packets
Loader – for Persistent State + Functions (Minimal Static Functionality)

Fig. 7. Abstract roles of SwitchWare subsystems.

packets provide light-weight, but very fined grained programmability. They basically serve as a scripting language that glues together node-resident services. An important advantage of this is that since node-resident services can now be composed, it makes sense for them to comprise smaller, more general, more reusable, pieces of functionality. Active extensions complement this by giving the system builder the ability to add new services dynamically and on-the-fly. Now, deploying a new protocol or service is just a matter of writing any new services that may be needed, while reusing existing pieces, and writing the simple packet programs that glue them together. This architecture has very significant flexibility advantages, but, as we will see, it also has an important safety and security benefit.

Despite the synergy between the two approaches in SwitchWare, much of the research focused on one approach or the other. Thus it is easiest to consider each thrust in turn, beginning with active extensions.

A. ALIEN

Active extensions allow the services of the network element to be dynamically extended, and thus their support is a fundamental question in node architecture. As noted in Section I, there is a design space with tradeoffs among flexibility, security, performance and usability. Thus, any AN node architecture must have a clear model of what regions of the design space it wishes to occupy, and what requirements will be placed on programmers of active extensions. The research goal of engineering an experimental system is to find desirable and perhaps unexpected local optima in the design space.

Work on active extensions in SwitchWare took place mainly in the context of the ALIEN system and its enhancements such as the secure active network environment (SANE). ALIEN was first prototyped in the Active Bridge [62], which showed that a complete bridge could be constructed "on-the-fly" from extensions that added buffered repeating, trivial local-area network (LAN) extension, self-learning and spanning tree functionalities. It also demonstrated that active extensions could be used to evolve the bridge from one spanning tree protocol to another and that, after failing a logical self-check, this evolution could be reversed. While the demonstration that active extensions could be used to build a functioning bridge lent credibility to the case for AN being useful, the more important architectural lesson was the understanding of the relationship between flexibility and security (in the form of privilege). ALIEN layered unprivileged programmable code over privileged programmable code, which in turn was layered over an immutable loader which served to load the privileged programmable code into the system.

A crucial design decision in ALIEN was its implementation in CAML [63]. CAML was chosen for a number of reasons. First, it was a strongly typed language with support for runtime safety checks and garbage collection. These features were crucial to our safety and security goals because they made it impossible for an extension to crash the system due to a pointer error or break security by using buffer overflow. Second, CAML was one of the few existing systems (Java being the other) to support remote dynamic loading of machine independent byte-code modules. This was crucial to our flexibility goals as it provided basic support for active extensions. Third, in CAML, when modules were dynamically loaded, they could be linked against a "thinned" module interface [64]. This allowed control of the functionality extensions could access, further contributing to our safety and security goals. Finally, we choose CAML over the single existing Java implementation for three reasons. First, CAML is an ML [65] family language. ML is a very well understood language from a PL theory point of view and our hope (ultimately unrealized) was that this would facilitate the use of formal methods to further assist with our safety and security goals. Second, at the time the CAML implementation was significantly faster than SUN's Java implementation, which contributed to our performance goals. Finally, there was considerable interest in the ML community [66] in networks as an area of study.

The use of programming language protections has advantages, as we have enumerated above, but extra mechanism is needed to trust these protections when the language system: 1) does not have exclusive access to the hardware; 2) must have protected channels to other, similar nodes; and 3) must control remote access to privileged resources. The diagram in Fig. 8 illustrates the detailed features of our solution, other details of which are described in Hicks, *et al.* [67].

The issue of exclusive access to the hardware has two implications. First, resource management of all types becomes challenging, as multiplexing of the hardware is carried out by another system, usually an operating system [68], [69]. Second, that system and all systems *it* in turn trusts must be trusted, recursively. To address the second of these two points, a secure bootstrap (AEGIS [70]) was developed to prevent subversion of our privilege enforcement from below,[3] we called the resulting system SANE [72], for "secure active network environment."

As we have mentioned before, one of the central contributions of the architecture was that we were able to define a privilege boundary *above* the immutable code of the loader. This allowed

[3]This focus on top-to-bottom integrity led to the discovery of new classes of security threats for programmable hardware [71].

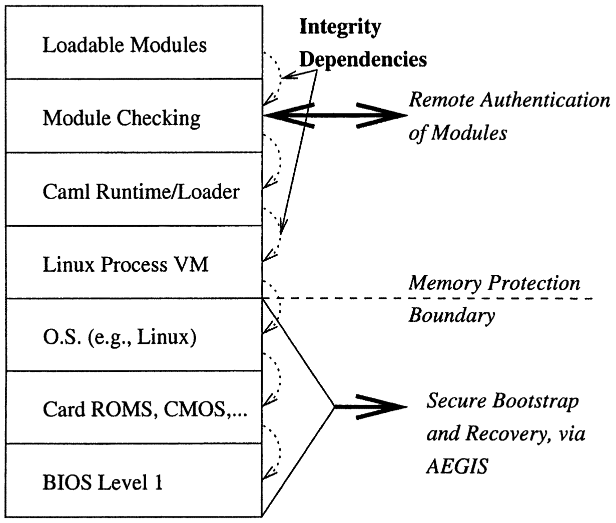

Fig. 8. Chaining layered integrity checks and node-node authen-tication in SANE.

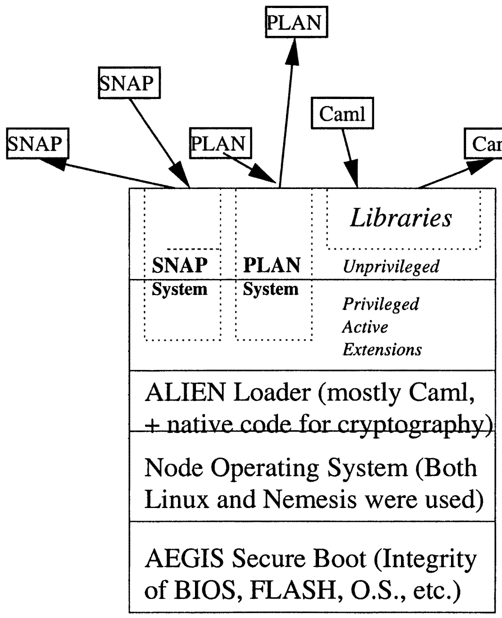
PLAN
SNAP
Caml
SNAP
PLAN
Caml
Libraries
SNAP System
PLAN System
Unprivileged
Privileged Active Extensions
ALIEN Loader (mostly Caml, + native code for cryptography)
Node Operating System (Both Linux and Nemesis were used)
AEGIS Secure Boot (Integrity of BIOS, FLASH, O.S., etc.)

Fig. 9. SwitchWare system architecture.

the loader to be extremely "thin," as hinted at in the illustration in Fig. 9.

While we initially used Linux as a development platform, it became clear that in the role of a network element operating system, even with secure bootstrap, Linux had severe limitations. In particular, it did not share the careful attention paid to security issues that we demanded of ALIEN itself, and the general Linux distributions had no support for metered resource con-trols and limits of any type, preventing ALIEN from supporting many applications which required timing and other resource guarantees.

The second thrust became resource guarantees, for resources such as memory, bandwidth and computation time. We addressed the resource control issues, including denial-of-service attacks [73] in one fashion in our secure quality of service handling [74] (SQoSH) architecture, and in another fashion entirely in the resource controlled active networking environment [75] (RCANE).

SQoSH used a new operating system called Piglet [76]–[78]. In the configuration in SQoSH, Piglet ran on one processor which managed networking activities, while Linux communicated data and control to and from Piglet via shared memory. Piglet provided enforcement of network resource allocation policies specified by the SANE system through the Linux kernel. Resource management was controlled with the same cryptographic credentialing system used to control code-loading in SANE, but adapted to the resource management interfaces offered by Piglet, thus integrating resource management with the other elements of the security architecture. However, SQoSH showed more that SANE could be used as a secure front end to a resource management mechanism, rather than the more crucial architectural result that SANE could be integrated with a complete resource management system, that included resources such as heap space.

The RCANE project protected active applications from "QoS crosstalk," where a lack of resource controls allowed applications to interfere with resource allocations of other active applications. RCANE was a collaboration with the University of Cambridge's Paul Menage, who melded the SwitchWare programming environment with a new multimedia operating system developed at the University of Cambridge Computer Laboratory named Nemesis [79]. Menage's work [80] examined many resource management issues, including garbage collection and CPU resources and developed a complete resource management architecture for SANE, which addressed quality of service (QoS) management, QoS crosstalk and was completely integrated with SwitchWare [75], including support for PLAN. This system demonstrated that with appropriate support, active extensions could operate with resource guarantees, enabling a wider range of network services and distributed applications.

B. *Packet Language for Active Networks*

Work on Active Packets in SwitchWare took place mainly in the context of the Packet Language for Active Networks (PLAN) system [81], [82]. PLAN is a *domain-specific language*. PLAN was tailored specifically to the task of acting as a "glue" language to compose operations provided by node services.

A critical aim in the PLAN design was to make PLAN packets fundamentally as lightweight as IP packets when performing similar functions. To meet this goal required that PLAN packets not be required to cryptographically authenticate before execution. However, it was also a goal that authentication be supported as an option, so that PLAN could perform privileged operations. (Related work in ALIEN using CAML for active packets showed that this was a good decision. CAML required authentication and thus proved unsuitable for light-weight operations.) To achieve this goal, PLAN's design drew explicitly from operating systems by creating a protection boundary between PLAN execution and invocations of service routines. That is service invocation plays the same role as a system call, service calls can be checked or even require authentication as is needed for the particular operation. This approach has proven to give the PLAN programmer a great deal of control over the cost of PLAN programs, significantly enhancing flexibility.

A number of other aspects of PLAN design drew from our experience in programming languages. From PL, we took the idea that PLAN should be strongly typed (it lacks dynamic storage allocation and arrays, so array bounds checking and garbage collection are unneeded). A novel aspect of the language is that although PLAN programmers may statically type check their programs before injecting them into the network, thus improving usability, PLAN programs are also dynamically type checked while being executed on a remote node. This guarantees that the node is protected, but avoids the cost of static checking when only a small part of the program is executed. Also from PL, we were motivated to design PLAN so that it might have a simple formal semantics. This required that the language itself be very simple and led to the eventual creation of an actual formal semantics [83]. In fact, Stehr and Talcott [84] even created a second specification in Maude. Another reason we wanted a simple language is that we believed that a simple restricted language would be easier to make security claims about and also make formal proofs of PLAN programs easier. A notable restriction is that PLAN programs are guaranteed to terminate and cannot loop infinitely. This means proofs need not demonstrate termination and gives us one useful bound on the resource usage of PLAN programs. It also means that PLAN is not as general a programming language as "Turing-complete" languages.

PLAN also drew from our experience in operating systems and distributed systems. The use of protection boundaries has already been noted, but another aspect is that PLAN does not support mutable data. This is important from an OS view because, if PLAN programs want to change the state of a node, they must call a service routine. That means that all issues of concurrency control become issues at the level of the service implementation (ALIEN in our case) and PLAN programs themselves are independent and can be executed independently. This is an important property for high performance packet processing. This immutable data property also played a role in the distributed systems nature of PLAN. By its very nature, PLAN is a distributed systems programming language. It uses the remote evaluation model for distributed computation, shipping both PLAN programs and data (in the form of function arguments) from node to node. Having all data be immutable means that it can be copied when it is transmitted and that the copied values have exactly the same meaning as the original. This is in contrast to RPC systems where copying a mutable value breaks the sharing relationship and changes the semantics of operations on the data.

We wanted to use PLAN to build a significant networking system, both to gain experience with our architecture and PLAN and to find any "holes" in PLAN's design. The system we built, PLANet [85], is a fully functioning internetwork. PLANet uses ALIEN to provide active extensions and PLAN for active packets and all packets are active. We felt that if we could implement internetworking, we could probably implement almost all other networking systems. This choice was fortuitous: we immediately recognized the need to support encapsulation as well as to treat packets as data, for purposes of fragmentation and encryption.

The solution we fixed upon [86], "chunks," elegantly combines the PL concept of "closure" and the remote-evaluation constructs used in PLAN. The idea is simple. A chunk is the function name to be called, the arguments to this function, and the code needed by the function. Chunks are an abstraction that captures the notion of "packet" in PLAN. Chunks can be sent to remote nodes where they can be executed, they can be treated as abstract data and passed as arguments to other chunks. This can be used to create generalized encapsulation, with PLAN execution driving demultiplexing. Finally chunks can be treated as arrays of bytes, allowing them to be encrypted, or split apart and reassembled. Chunks make it easy to create protocol boosters that are inserted and deleted even at a packet by packet level. Chunks represent perhaps the first new abstraction in active packets since Wall's [1] work.

C. Further Developments

PLAN, PLANet, and ALIEN eventually led to work on several "second generation" AN systems, which built on previous SwitchWare experience.

The first such system is Michael Hick's thesis work on dynamic updating [87], [88]. While experimenting with network service level evolution in PLANet, it became clear that the dynamic loading supported by CAML required that interfaces would need to be designed for evolution, not just evolved when the need arose. Dynamic updating addresses this issue by allowing almost arbitrary updates to be made to running code, including changing data representations on-the-fly. Dynamic updating was implemented as part of the typed assembly language (TAL) system [89]. TAL is one of the newest approaches to providing safe, mobile code.

The second such system is Jon Moore's thesis work on Safe and nimble active packets (SNAP) [90], [91]. SNAP draws lessons from many first-generation active packet systems, including ANTS [9] and Smart Packets [51], but it is most closely related to PLAN. The two major goals of SNAP were to improve performance and to make active packets safer with regards to resource allocation and use. To achieve better performance, SNAP was designed to be a lower-level (byte-code) language than PLAN and its design allows for much simpler memory management and significant improvements in marshaling and unmarshalling costs. SNAP was implemented in the Linux kernel. SNAP meet its performance goal and just slightly slower (a few percent) than the Linux IP implementation. To address resource use, SNAP introduces a model in which each packet can use only a linear amount of CPU, memory, and bandwidth, each time it executes on a node. This is a very severe restriction and to implement it required limiting the byte-code language to making only forward branches. Thus SNAP goes much further than PLAN in using language restrictions to gain safety and security. Experiments with SNAP and the PLAN-to-SNAP compiler [92] suggest that SNAP still retains enough flexibility to be generally useful.

The architecture of the complete system is shown in Fig. 9.

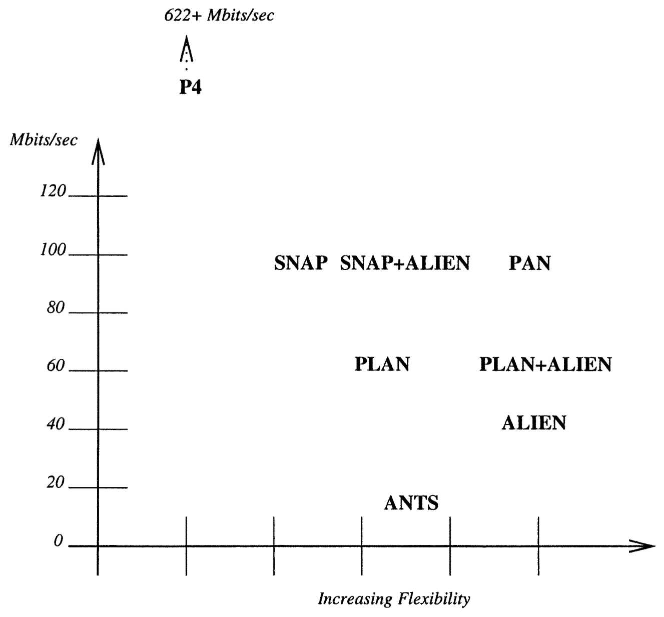

Fig. 10. Flexibility versus measured performance tradeoffs.

D. Tradeoffs

Fig. 10 illustrates results in the active networking design (sub)space of performance versus flexibility tradeoffs. The P4 alone, while operational at 155 Mb/s and workable to 622+ Mbits/s [47] with different component choices, is standalone the least flexible of the systems, as it can only accommodate "programs" which can be expressed within the constraints of a field-programmable logic device. SNAP also has more limited flexibility, but can operate at 100 Mb/s. PLAN is more flexible than SNAP, while ANTS can support nonterminating execution, making it more flexible than PLAN. ALIEN is the only general-purpose Active extension system shown. Its flexibility is high since almost arbitrary changes can be made in the underlying node. ALIEN's performance depends on whether it is using CAML-based Active packets, or ones based on PLAN or SNAP. There is no SNAP+ALIEN, but it is reasonable to assume it would be similar to SNAP alone. We also show practical active network (PAN). As essentially a kernel module loader, PAN performs very well and is of course quite flexible, but no better than SNAP+ALIEN.

What PAN lacks, of course, is any security. If we plotted an additional dimension of the design space, we would find the most secure combination would be the combination of ALIEN and SNAP, following our experience with PLAN+ALIEN. This gives resource bounds at the active packet level, and with the addition of some operating system support, such as SQoSH's Piglet or RCANE's Nemesis, at the active extension level as well. Problems with the Java virtual machine security, plus the lack of resource management and support for formal methods suggests that ANTS is less secure than PLAN or SNAP, particularly in the latter case where SNAP's resource usage is linear in the size of the active packet.

E. Summary

The performance of AN systems is adequate for the network edge, as discussed above in Section VI-D and compared against

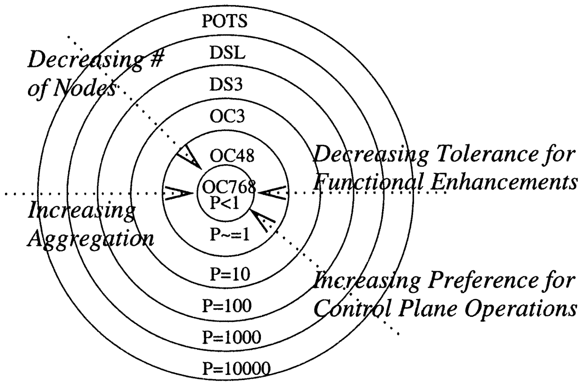
POTS
DSL
DS3
OC3
OC48
OC768
P<1
P~=1
P=10
P=100
P=1000
P=10000
Decreasing # of Nodes
Increasing Aggregation
Decreasing Tolerance for Functional Enhancements
Increasing Preference for Control Plane Operations

Fig. 11. Computation versus network performance tradeoffs.

Fig. 11, which shows the number of programmed instructions (P) that can be executed by a general purpose processor on a small data unit such as a bit, byte or word without delaying the data transmission, along with some design pressures that follow from the value of P. Specifically, almost all access networks have bandwidths less than 100 Mbits/s, a speed at which many of the systems described in this paper operate at, or near to (see Fig. 10). The P4 system described in Section IV can operate at or near line rates since it is implemented using programmable hardware; current components would allow operation above 1 Gbps for functions realizable in field programmable gate array technology.

Additionally, new *network processors* [93]–[95] are offering a path to wire-speed performance, with specialized architectures suited to concurrent execution of networking operations. The network processor technology is positioned to provide powerful programming environments to network element developers.

The SwitchWare node architecture addressed all of the fundamental problems of security [61], [72], [96], [97]: controlling integrity, resource multiplexing and management, and authentication. The security solutions were portable and are useful today in many contexts, from loading code onto phones to providing support for multiple isolated execution environments.

VII. Future Directions

Active networking and AN-inspired approaches continue their transformation of networking. In this section, we first review three areas that both benefit from lessons learned in active networking, and might well absorb more lessons from the tradeoffs discovered among flexibility, performance, usability and security. We then consider the implications of the Internet Engineering Task Force's (IETF) study of forwarding and control element separation (ForCES) and opportunities it creates for AN. Finally, we discuss active router control, an approach to exploiting the role separation creates for AN-based control elements in the Internet.

A. Middleboxes, Overlays, and Sensor Networks

Middleboxes: Our observations suggest that points at or near the IP network's "edge" (rather than the "core," which seems likely to evolve toward optical circuit-switching in any case [98]) will be the deployment point for active networking technologies. Not surprisingly, this is already happening: domain-specific "middleboxes" [99] are appearing there such as firewalls, network address translators (NATs) and intrusion detection systems (IDSs). To quote from the request for comments (RFC),

> . . .Instead of concentrating diversity and function at the end systems, they spread diversity and function throughout the network.

Examples given of such middleboxes include

- NATs [100]–[103];
- NAT with protocol translator (NAT-PT) [104];
- SOCKS gateway [105];
- Packet classifiers, markers and schedulers [106];
- Transmission control protocol (TCP) performance enhancing proxies [107];
- IP firewalls [108], [109];
- Gatekeepers/session control boxes [110], [111];
- Transcoders;
- Proxies [110], [112].

The opportunity here is clear; these various application-driven "ad hoc" examples of STF functionality can be unified in a common framework, and embedded in a common programming model reinforced with the safety and security lessons learned from active networking.

Overlay Networks: The current focus on peer-to-peer systems [113] is a way to introduce new services using an overlay, in the style of the World Wide Web. "Peers" use the IP network layer as a "link;"each peer serves as a "router" in the network. Since these peers are fully programmable nodes, essentially arbitrary distributed computing architectures can be constructed; the approach has mainly been applied to data and file services (e.g., the music distribution of Napster and Gnutella) as these are not particularly sensitive to latencies and require restricted access to machine features. Of course, more aggressive approaches are possible when considering network services. Some early overlay work in active networking was done by Crowcroft [114] and his collaborators at University College, London, resulting in, for example, in an architecture for application layer routing [115]. Other early systems such as Emulab [116] and X-Bone [117] illustrated the design space, stimulating larger-scale experiments[4] such as PlanetLab [118]. Perhaps most interestingly, some of these infrastructures [119] are overlay equivalents of the STF model introduced earlier in this paper.

The overlay approach avoids altering the behavior of the network layer in introducing new services [120], at some cost in ignorance of relevant network layer characteristics such as topology, multiplexing and latency. As the network layer has proven difficult to change (except through interposition of middleboxes, as discussed above) this approach avoids the difficulty and lets experimentation proceed. If peer-to-peer systems become sufficiently successful, they may stimulate selective reshaping of the network layer from above [118], in

[4]Interestingly, Crowcroft *et al.*'s ALAN software is was used with X-Bone and is now used with PlanetLab.

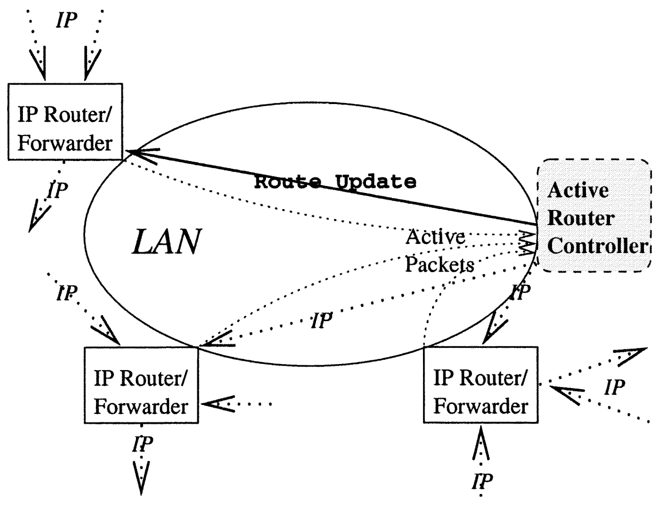

Fig. 12. Active router controller (ARC) managing a set of forwarder/routers.

the same way that IP, initially a software overlay, reshaped router hardware.

Sensor Networks: The availability of cheap powerful sensors with communications capability has stimulated interest in access to, and organization of, groups of these sensors [121]. The networking issues are interesting, as the devices are often low-powered, with limited communications capability (especially, low bandwidth) and limited processing capacity. They may also be unreliable. Thus, they are typically organized as "ad-hoc networks," where the devices themselves self-organize into a network. Applications which use these networks are often more interested in the aggregate information from a group rather than data from a particular sensor, and this aggregation of information can result in reductions in data traffic.

B. ForCES Augmenting the Internet

The IETF's forwarding and control element separation (ForCES) working group [122] models router architectures in a way that allows introduction of programmable and active technologies into the Internet control plane, in the style of BBN's FIRE [123].

The principal observation we make is that forwarding and routing are distinct activities. Routing is part of the "control plane" of the IP Internet, while forwarding is the "transport plane." These activities were consolidated in traditional IP routers, but are now recognized as logically separate (viz. MPLS).

The separation permits more general network elements to replace IP routers in performing control plane activities, allowing distributed routing with improved performance, slightly more global optimization, and perhaps surprisingly, an increase in security. In addition, the separation of routing and forwarding functions permits decoupling their performance, allowing better tracking of technology improvements in both forwarder mechanics and routing algorithms.

C. Active Router Control

Active router control uses fast forwarders as a virtual link layer, managed by specialized active router controllers (ARCs). An illustration of this idea is shown in Fig. 12. Using a set of routers as a "router in a room" is not uncommon; one could simply reduce their autonomy and specialize them to IP forwarding—making way for source routing over all-optical networks or routing/router control internal to the network. The use of general purpose network elements permits this separation of concerns, while offering the *potential* for improvement of the Internet on a number of axes.

Fig. 12 illustrates an ARC, perhaps modeled after the active bridge [62], colocated on a fast LAN with a set of router/forwarders. In this configuration, choices of routes are made by the ARC. This can be done with unmodified routers via command interfaces, or more easily if the routers are architected to be dumb forwarders slaved to a local route controller. ARC intercommunication is done via IP packets; both "in-band" and "out-of-band" (private link) models can be used. ARC operations across the LAN will incur a delay relative to operations internal to an integrated router/forwarder, but given the control/flow distinction between routing and forwarding, it is not a major limitation of the design. A significant advantage is that specialized forwarding tables can be loaded into each forwarder; these tables can be small since entries must exist only for adjacent nodes.

A key advantage of the ARC model is that for computationally-centered tasks (routing, or more general computations if the programmability of ARCs is exploited), a computer which tracks computer technology trend exponentials (faster CPU, larger RAM) is used, while the forwarders independently track networking technology trend exponentials such as bandwidth improvements.[5] While a potential weakness of the scheme is ARC failures, this is both unlikely and can be addressed through classical techniques such as heartbeats, redundancy, etc. An apparent risk is that a particular route carrying routing information from an ARC to a forwarder may break, but this risk is never worse than in the present Internet, and discovery of link failures can fail forwarders over to a new ARC if necessary. ARCs initialize by detecting adjacent forwarders,

[5]Not surprisingly, the highest performance IP routers have also adopted such a distributed architecture, although with different goals and roles for the line cards.

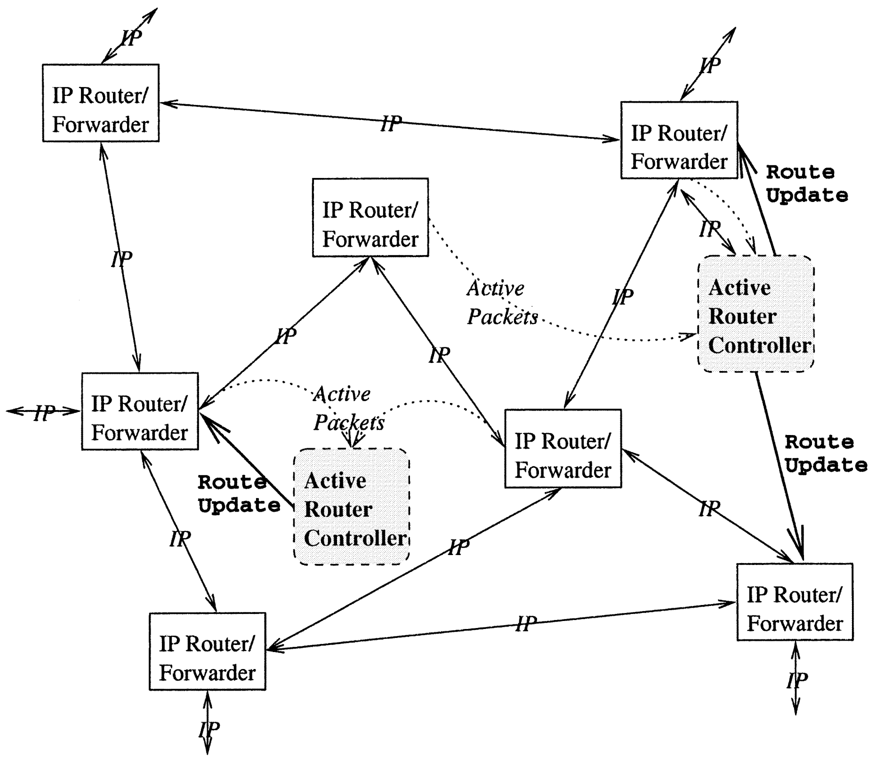

Fig. 13. ARCs distributed throughout an Internet.

using IP packets to extend control to other forwarders. ARCs intercommunicate using IP packets, essentially using IP as a universal link layer.

The basic distributed control architecture of Fig. 12 can be replicated throughout an Internetwork, with the active elements using the managed Internet routes as link layers. This is shown in Fig. 13, where a set of active nodes have been grafted into a larger collection of forwarders to create an active Internet.

VIII. Conclusion

The AN research program has had effects on the field of networking, both broad and deep. Most all of the initial questions raised about active networking, particular those of performance and security risks, have been addressed. Further, technology transfer and commercialization of AN or specializations of it has been widespread and vibrant communities have developed worldwide. Finally, we suggest, as we observed in Section VII, that there is far more we can do.

Worldwide efforts in active and programmable networking are underway, particularly in Europe (see, e.g., the "Future Active IP Network" project [124] or the ALPINE project [114]). In addition to the network processor market, other efforts are seeking to commercialize one or more aspects of active networking, and active networking inspired approaches such as "packet-marking" [125], [126] are penetrating products.

The wireless industry is also absorbing active network approaches, ranging from mobile telephony industry's use of mobile code to the software definition of radio [127], [128].

The need for flexible network evolution remains, and in many ways the needs of distributed computing are still not sufficiently well met by the exclusion of *computing* application programmer interface (APIs) from the dominant network infrastructure.

The uptake on active networking has been slow, and its near-term impact has not been sufficient, but the marketplace is delivering network embedded programmability, albeit in an ad-hoc and purpose-built fashion. For example, as we noted in Section VII, NAT boxes are simple STF-style translators, and firewalls are event-driven programs with simple actions (`pass`, `drop`, or `log`) driven by complex rule specifications in firewall-specific rule expression languages.

Perhaps we should just treat this as a case of "le mieux est l'ennemi du bien" [129] and move on, but the difficulty with the ad hoc approach is that these purpose-built systems cannot absorb modules and programming styles from existing systems, and are difficult to extend and compose. The active networking approach provides a common infrastructure with which such systems can be built in a robust and secure manner. The IETF "middlebox" work [99] suggests that the value of a unified programming model for network services is increasing, rather than decreasing.

We pose three open questions.

- How can an operational network system evolve *piecewise*, so as not to incur negative economic effects on early adopters of the technology?
- What knowledge about the network is necessary to diagnose problems, drive reconfiguration of the network architecture, etc.?
- How can one automate the process of choosing protocol elements, such as protocol boosters, to compose an optimal protocol for a given application and network conditions?

ANs offer an opportunity to build a truly flexible distributed computing infrastructure, where the programmer is in command of all aspects of their distributed computing model.

Acknowledgment

The authors wish to particularly thank D. Maughan of DARPA for his energetic support and insightful guidance of the DARPA AN program. W. D. Sincoskie's questions stimulated the "store-translate-and-forward" approach. Collaboration with D. Feldmeier on Protocol Boosters provided some first insights into how to get STF right, and we continued to work together on Active Networking in the SwitchWare project. B. Lyles pointed out the visionary work of Zander and Forchheimer in the Softnet project, and B. Plattner pointed out the work done by Wall in his analysis of messages as active agents.

D. S. Alexander led the development of our first prototype active networking system, and suggested bridging as a convincing application for on-the-fly construction of a software element. M. Shaw contributed to this implementation and its evaluation. W. Arbaugh and A. Keromytis figured out how to secure the prototype system as it evolved into Alexander's ALIEN system. I. Hadzic designed and implemented the P4, showing that programmable infrastructures could operate at line speeds, and in many ways foresaw network processors. S. Muir and J. Shapiro designed novel alternative operating systems architectures (Piglet and EROS, respectively) to understand the resource control and protection design space. M. Hicks, P. Kakkar, and J. Moore were instrumental in the initial PLAN design and implementation. M. Hicks and J. Moore also were active in the design of PLANet and are the implementors of that system. M. Hicks developed the dynamic software updating system, for which he won the 2002 SIGPLAN Best Programming Language Dissertation Award. Finally, J. Moore developed the SNAP system.

Collaborators at Penn in the SwitchWare project, particularly D. Farber and C. Gunter, helped crystallize design decisions and C. Gunter was a principal contributor to PLAN and assisted with the design of PLANet. W. Marcus, M. Segal and A. Bogovic at Bellcore (now Telcordia Technologies) were significant collaborators, implementing a substantial protocol boosters infrastructure and a variety of applications for active networks, including a powerful publish/subscribe information distribution system.

G. Minden had the vision to see that Protocol Boosters were a path to a new form of network/computer integration, enabling an improved binding between networks and distributed applications. D. Tennenhouse, first at MIT and then at DARPA, championed the DARPA Active Networks program into existence. The work of Tennenhouse, J. Guttag, and D. Wetherall, resulted in the most widely used Active Networking system, ANTS.

Many people have reviewed, and improved, earlier versions of this paper with their comments on both history and the description of it in this paper. These include W. D. Sincoskie, D. Feldmeier, D. S. Alexander, B. Plattner, C. Partridge, I. Hadzic, G. Minden, A. Keromytis, D. Farber, S. da Silva, J. Crowcroft, M. Hicks, S. Denazis, W. Marcus, J. Davin, D. Wetherall, S. Shenker, C. Tschudin, L. Peterson, D. Clark and R. Braden. Any remaining inaccuracies are ours alone.

References

[1] D. W. Wall, "Messages as active agents," in *Proc. 9th Annu. POPL*, 1982, pp. 34–39.
[2] J. Zander and R. Forchheimer, "Preliminary Specification of a Distributed Packet Radio System Using the Amateur Band," Univ. Linköping, Linköping, Sweden, Jan. 1980.
[3] ——, "Softnet—An approach to higher level packet radio," in *Proc. AMRAD Conf.*, 1983.
[4] J. M. Smith, "Reflections on Active Networking," Dept. Comput. Inform. Sci., Univ. Pennsylvania, Tech. Rep. MS-CIS-03-05, 2003.
[5] D. J. Farber and K. Larson, "The architecture of a distributed computer system—An informal description," Inform. Comput. Sci., Univ. California, Irvine, Tech. Rep. 11, 1970.
[6] D. J. Farber, "The distributed computing system," in *Proc. COMPCOM*, 1973, pp. 31–34.
[7] J. M. Smith, "A survey of process migration mechanisms," in *Proc. ACM SIGOPS Oper. Syst. Rev.*, July 1988, pp. 28–40.
[8] J. M. Smith and J. Ioannidis, "Implementing remote fork() with checkpoint/restart," *IEEE Tech. Committee Oper. Syst. Newsletter*, pp. 12–16, Feb. 1989.
[9] D. J. Wetherall, J. Guttag, and D. L. Tennenhouse, "ANTS: Atoolkit for building and dynamically deploying network protocols," in *Proc. IEEE OpenArch*, 1998, pp. 117–129.
[10] B. J. Nelson, "Remote Procedure Call," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1981.
[11] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, no. 1, pp. 39–59, Feb. 1984.
[12] J. Gosling, D. S. H. Rosenthal, and M. J. Arden, *The NeWS Book: An Introduction to the Network/Extensible Window System*. New York: Springer-Verlag, 1989.
[13] J. W. Stamos, "Remote Evaluation," Ph.D. dissertation, Mass. Inst. Technol., Cambridge, Jan. 1986.
[14] J. W. Stamos and D. K. Gifford, "Remote evaluation," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 4, pp. 537–565, Oct. 1990.
[15] ——, "Implementing remote evaluation," *IEEE Trans. Softw. Eng.*, vol. 16, pp. 710–722, July 1990.
[16] S. M. Clamen, L. D. Leibengood, S. M. Nettles, and J. M. Wing, "Reliable distributed computing with avalon/common lisp," in *Proc. Int. Conf. Computer Languages*, 1990, pp. 169–179.
[17] J. M. Smith, J. Gerald, and Q. Maguire, "Process migration: effects on scientific computation," *ACM SIGPLAN Notices*, vol. 23, no. 3, pp. 102–106, Mar. 1988.
[18] "Gigabit network testbeds," *IEEE Comput.*, vol. 23, pp. 77–80, Sept. 1990.
[19] J. Babcock, "SONET: A practical perspective," *Bus. Commun. Rev.*, vol. 20, no. 9, pp. 59–63, Sept. 1990.
[20] "Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria," Bellcore, Red Bank, NJ, Report TR-NWT-000 253, Dec. 1991.
[21] Gigabit Rate Transmit Receive Chip Set Technical Data, Hewlett-Packard, 1992.
[22] HP9000 Series 700 HIPPI Interface HP J2069A, Hewlett-Packard, 1994.
[23] W. St. John and D. DuBois, "HiPPI-SONET gateway," in *CASA Gigabit Testbed Annual Rep.*, 1993, pp. 47–52.
[24] J. Giacopelli, J. Hickey, W. Marcus, W. D. Sincoskie, and M. Littlewood, "Sunshine: A high-performance self-routing broadband packet switch architecture," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1289–1298, Oct. 1991.
[25] T. Bogovic, B. Davie, J. Hickey, W. Marcus, V. Massa, L. Trajkovic, and D. Wilson, "The architecture of the sunshine broadband testbed," in *Proc. of XIV Internation Switching Symp.*, Oct. 1992, pp. 204–208.
[26] D. Clark, B. Davie, D. Farber, I. Gopal, B. Kadaba, W. Sincoskie, J. Smith, and D. Tennenhouse, "An overview of the AURORA gigabit testbed," in *Proc. IEEE INFOCOM Conf.*, 1992, pp. 569–581.
[27] D. D. Clark *et al.*, "The AURORA gigabit testbed," *Computer Netw. ISDN Syst.*, vol. 25, no. 6, pp. 599–621, Jan. 1993.
[28] W. D. Sincoskie, "Broadband packet switching: a personal perspective," *IEEE Commun. Mag.*, vol. 40, pp. 54–66, July 2002.
[29] P. Newman, G. Minshall, and T. Lyon, "IP switching—ATM under IP," *IEEE/ACM Trans. Networking*, pp. 117–129, Apr. 1998.
[30] T. Li, "MPLS and the evolving internet architecture," *IEEE Commun. Mag.*, vol. 37, pp. 38–41, Dec. 1999.
[31] D. D. Clark, "The design philosophy of the DARPA internet protocols," *ACM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 106–114, Aug. 1988.
[32] A. S. Tanenbaum, *Computer Networks*, 2nd ed: Prentice-Hall, 1988.
[33] J. E. van der Merwe and I. M. Leslie, "Switchlets and dynamic virtual ATM networks," in *Proc. IFIP Integrated Network Management V*, May 1997, pp. 355–368.

[34] S. Rooney, "Connection closures: adding application-defined behavior to network connections," *ACM Comput. Commun. Rev.*, vol. 27, no. 2, pp. 74–88, 1997.

[35] D. A. Halls and S. G. Rooney, "Controlling the tempest: adaptive management in advanced ATM control architectures," *IEEE J. Select. Areas Commun.*, vol. 16, Apr. 1998.

[36] J. E. van der Merwe and I. M. Leslie, "Service-specific control architectures for ATM," *IEEE J. Select. Areas Commun.*, vol. 16, Apr. 1998.

[37] W. D. Sincoskie, Personal Communications on Interservice Networking, 1992.

[38] J. M. Smith, "Store, Translate and Forward (STF) Networks," unpublished, 1993.

[39] N. C. Hutchinson and L. L. Peterson, "The x-Kernel: an architecture for implementing network protocols," *IEEE Trans. Softw. Eng.*, vol. 17, pp. 64–76, Jan. 1991.

[40] D. Ritchie, "A stream input-output system," *AT&T Bell Lab. Tech. J.*, pt. 2, vol. 63, no. 8, pp. 1897–1910, Oct. 1984.

[41] C. Tschudin, "Flexible protocol stacks," in *Proc. ACM SIGCOMM Conf.*, Aug. 1991, pp. 197–204.

[42] T. Plagemann, B. Plattner, M. Vogt, and T. Walter, "A model for dynamic configuration of light-weight protocols," in *Proc. 3rd IEEE Workshop Future Trends Distributed Systems*, Apr. 1992, pp. 100–106.

[43] A. Mallet, J. D. Chung, and J. M. Smith, "Operating systems support for protocol boosters," in *Proc. HIPPARCH Workshop*, June 1997.

[44] W. S. Marcus, A. J. McAuley, and T. Raleigh, "Protocol boosters: a kernel-level implementation," in *Proc. IEEE GLOBECOM*, Nov. 1998, pp. 1619–1623.

[45] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus, and T. M. Raleigh, "Protocol boosters," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 437–444, Apr. 1998.

[46] I. Hadžić and J. M. Smith, "P4: A platform for FPGA implementation of protocol boosters," in *Field Programmable Logic 1997*. New York: Springer-Verlag, 1997, ser. Lecture Notes in Computer Science, no. 1304, pp. 438–447.

[47] I. Hadžić, "Applying Reconfigurable Computing to Reconfigurable Networks," Ph.D. dissertation, Univ. Pennsylvania, Philadelphia, Sept. 1999.

[48] Y. Yemini and S. daSilva, "Toward programmable networks," in *IFIP/IEEE Int. Workshop Distributed Systems: Operations Management*, Oct. 1996.

[49] J. Hartman, U. Manber, L. Peterson, and T. Proebsting, "Liquid Software: A New Paradigm for Networked Systems," Univ. Arizona, Tuscon, AZ, Tech. Rep. 96-11, June 1996.

[50] C. Partridge and A. Jackson. (1996) Smart Packets. BBN. [Online] Available: http://www.net-tech.bbn.com/smtpkts/smtpkts-index.html

[51] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, D. Rockwell, and C. Partridge, "Smart packets for active networks," *ACM Trans. Comput. Syst.*, vol. 18, no. 1, pp. 67–88, Feb. 2000.

[52] D. Tennenhouse and D. Wetherall, "Toward an active network architecture," *Comput. Commun. Rev.*, vol. 26, no. 2, 1996.

[53] J. M. Smith, D. J. Farber, C. A. Gunter, S. M. Nettles, D. C. Feldmeier, and W. D. Sincoskie. (1996, June) SwitchWare: Accelerating Network Evolution (White Paper). Univ. Pennsylvania, Philadelphia. [Online]Available: http//www.cis.upenn.edu/jms/white-paper.ps

[54] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Commun.*, vol. 35, pp. 80–86, Jan. 1997.

[55] M. Hicks, J. T. Moore, D. Wetherall, and S. Nettles, "Experiences with capsule-based active networking," in *Proc. IEEE DARPA Active Networks Conference and Exposition (DANCE)*, May 2002, pp. 16–24.

[56] R. Milner, M. Tofte, and R. Harper, *The Definition of Standard ML*. Cambridge, MA: MIT Press, 1990.

[57] J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*. Reading, MA: Addison-Wesley, 1996.

[58] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proc. 17th Symp. Security 17 Privacy*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1996, pp. 164–173.

[59] Switchware Home Page. [Online] Available: http://www.cis.upenn.edu/switchware

[60] J. M. Smith, D. J. Farber, D. C. Feldmeier, C. A. Gunter, S. M. Nettles, W. D. Sincoskie, and S. Alexander, "Switchware: Accelerating Network Evolution," Dept. Comput. Inform. Sci., Univ. Pennsylvania, Philadelphia, Tech. Rep. MS-CIS-96-38, 1996.

[61] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, "The SwitchWare active network architecture," *IEEE Network*, pp. 29–36, May/June 1998.

[62] D. S. Alexander, M. Shaw, S. Nettles, and J. Smith, "Active bridging," in *Proc. ACM SIGCOMM Conf.*, 1997, pp. 101–111.

[63] X. Leroy, *The CAML Special Light System (Release 1.10)*. Rocquencourt, France: INRIA, Nov. 1995.

[64] F. Louaix, "A web navigator with applets in CAML," in *Proc. 5th WWW Conf.*, 1996, pp. 1365–1371.

[65] R. Milner, M. Tofte, and R. Harper, *The Definition of Standard ML*. Cambridge, MA: MIT Press, 1990.

[66] E. Biagioni, "A structured TCP in standard ML," in *Proc. 1994 SIGCOMM Conf.*, 1994, pp. 36–45.

[67] M. W. Hicks, A. D. Keromytis, and J. M. Smith, "A secure PLAN," *IEEE Trans. Syst., Man, Cybern. C*, vol. 33, pp. 413–426, Aug. 2003.

[68] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullman, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, and J. Hartman, "An OS interface for active routers," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 473–487, Mar. 2001.

[69] P. Tullmann, M. Hibler, and J. Lepreau, "Janos: A java-oriented OS for active network nodes," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 501–510, Mar. 2001.

[70] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Proc. IEEE Security Privacy Conf.*, May 1997, pp. 65–71.

[71] I. Hadžić, S. Udani, and J. M. Smith, "FPGA viruses," in *Proc. 9th Int. Workshop Field-Programmable Logic Applications, FPL'99*, Aug. 1999.

[72] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, "A secure active network environment architecture: Realization in switchWare," *IEEE Network*, pp. 37–45, May/June 1998.

[73] R. M. Needham, "Denial of service: an example," *Commun. ACM*, vol. 37, no. 11, pp. 42–46, Nov. 1994.

[74] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, S. Muir, and J. M. Smith, "Secure quality of service handling (SQoSH)," *IEEE Commun.*, vol. 38, pp. 106–112, Apr. 2000.

[75] D. Alexander, P. Menage, A. Keromytis, W. Arbaugh, K. Anagnostakis, and J. Smith, "The price of safety in an active network," *J. Commun.*, vol. 3, no. 1, pp. 4–18, Mar. 2001.

[76] S. J. Muir and J. M. Smith, "AsyMOS: An asymmetric multiprocessor operating system," in *Proc. 1st OpenARCH Conf.*, Apr. 1998, pp. 25–34.

[77] ——, "Supporting continuous media in the piglet OS," in *Proc. 8th Int. Workshop Network Operating Systems Support Digital Audio Video*, July 1998, pp. 99–102.

[78] S. J. Muir, "Piglet: An Operating System for Network Appliances," Ph.D. dissertation, CIS Dept., Univ. Pennsylvania, Philadelphia, 2001.

[79] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1280–1297, Sept. 1996.

[80] P. B. Menage, "Resource Control of Untrusted Code in an Open Programmable Network," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., 2000.

[81] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles, "PLAN: A Packet Language for Active Networks," in *Proc. Int. Conf. Functional Programming*, 1998, pp. 86–93.

[82] Plan Home Page. [Online] Available: http://www.cis.upenn.edu/switchware/PLAN

[83] P. Kakkar, M. Hicks, J. T. Moore, and C. A. Gunter, "Specifying the PLAN network programming language," *Electron. Notes Theoret. Comput. Sci.*, Sept. 1999.

[84] M.-O. Stehr and C. Talcott, "Plan in maude: Specifying an active network programming language," in *Proc. 4th Int. Workshop Rewriting Logic Its Applications*. New York: Elsevier, Sept. 19–21, 2002, vol. 71, ser. Electron. Notes Theoret. Comput. Sci..

[85] M. Hicks, J. T. Moore, D. S. Alexander, C. A. Gunter, and S. Nettles, "PLANet: An active internetwork," in *Proc. 18th IEEE Computer Communication Society INFOCOM Conf.*: IEEE, 1999, pp. 1124–1133.

[86] J. T. Moore, M. Hicks, and S. Nettles, "Chunks in PLAN: Language support for programs as packets," in *Proc. 37th Annu. Allerton Conf. Communication, Control, Computing*, Sept. 1999.

[87] M. W. Hicks, J. T. Moore, and S. Nettles, "Dynamic software updating," in *SIGPLAN Conf. Programming Language Design Implementation*, 2001, pp. 13–23.

[88] M. Hicks, "Dynamic Software Updating," Ph.D. dissertation, CIS Dept., Univ. Pennsylvania, Philadelphia, 2001.

[89] G. Morrisett, D. Walker, K. Crary, and N. Glew, "From system F to typed assembly language," in *Proc. 25th ACM Symp. Principles of Programming Languages*, Jan. 1998, pp. 85–97.

[90] J. T. Moore, M. Hicks, and S. Nettles, "Practical programmable packets," in *Proc. 20th IEEE Computer Communication Soc. INFOCOM Conf.*: IEEE, Apr. 2001, pp. 41–50.
[91] J. T. Moore, "Practical Active Packets," Ph.D. dissertation, Univ. Pennsylvania, Philadelphia, Sept. 2002.
[92] M. Hicks, J. T. Moore, and S. Nettles, "Compiling PLAN to SNAP," in *Proc. 3rd Int. Working Conf. Active Networks*, vol. 2207, ser. Lecture Notes in Computer Science, I. W. Marshall, S. Nettles, and N. Wakamiya, Eds., Oct. 2001, pp. 134–151.
[93] Intel IXP Architecture Network Processors. [Online]. Available: http://www.intel.com/design/network/products/npfamily/
[94] IBM PowerNP Network Processors. [Online]. Available: http://www-3.ibm.com/chips/products/wired/products/network processors.html
[95] Agere Network Processors. [Online]. Available: http://www.agere.com/enter-prise metro access/network processors.html
[96] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, "Safety and security of programmable network infrastructures," *IEEE Commun.*, vol. 36, pp. 84–92, 1998.
[97] ——, "Security in active networks," in *Secure Internet Programming*. New York: Springer-Verlag, 1999, ser. Lecture Notes in Computer Science, Berlin, Germany, pp. 433–451.
[98] P. Molinero-Fernandez, N. McKeown, and H. Zhang, "Is IP going to take over the world (of communications)?," in *Proc. ACMCC*, Jan. 2003, pp. 113–117.
[99] B. Carpenter and S. Brim, Middleboxes: Taxonomy and Issues, Internet Engineering Task Force, RFC 3234, Available: [Online] www.iets.org, Feb. 2002.
[100] P. Srisuresh and M. Holdrege, IP Network Address Translator (NAT) Terminology and Considerations, Aug. 1999.
[101] T. Hain, Architectural Implications of NAT, Internet RFC 2993, Available: [Online] www.iets.org, Nov. 2000.
[102] P. Srisuresh and K. Egevang, Traditional IP Network Address Translator (Traditional NAT), Internet RFC 3022, [Online] Available: www.iets.org, Jan. 2001.
[103] M. Holdrege and P. Srisuresh, Protocol Complications With the IP Network Address Translator, Internet RFC 3027, Available: [Online] www.iets.org, Jan. 2001.
[104] G. Tsirtsis and P. Srisuresh, Network Address Translation—Protocol Translation (NAT-PT), Internet RFC 2766, [Online] Available: www.iets.org, Feb. 2000.
[105] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, SOCKS Protocol Version 5, Internet RFC 1928, [Online] Available: www.iets.org, March 1996.
[106] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An Architecture for Differentiated Service, Internet RFC 2475, [Online] Available: www.iets.org, Dec. 1998.
[107] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations, Internet RFC 3135, [Online] Available: www.iets.org, June 2001.
[108] N. Freed, Behavior of and Requirements for Internet Fire-Walls, Internet RFC 2979, [Online] Available: www.iets.org, Oct. 2000.
[109] B. Cheswick and S. Bellovin, *Firewalls and Internet Security: Repelling the Wily Hacker*. Reading, MA: Addison-Wesley, 1994.
[110] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, SIP: Session Initiation Protocol, Internet RFC 2543, [Online] Available: www.iets.org, Mar. 1999.
[111] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, and J. Segers, Megaco Protocol 1.0, Internet RFC 3015, [Online] Available: www.iets.org, Nov. 2000.
[112] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, Hypertext Transfer Protocol—HTTP/1.1, Internet RFC 2616, [Online] Available: www.iets.org, June 1999.
[113] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: Ascalable peer-to-peer lookup protocol for internet applications," in *Proc. ACM SIGCOMM Conf.*, 2001, pp. 149–160.
[114] Application Level Programmable Inter-Network Environment Project Web Page, http://www.cs.ucl.ac.uk/alpine/. [Online]
[115] A. Ghosh, M. Fry, and J. Crowcroft, "An architecture for application layer routing," in *Proc. 2nd Int. Working Conf. Active Networks*, vol. 1942, ser. Lecture Notes in Computer Science, H. Yashuda, Ed., Oct. 2000, pp. 71–86.
[116] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, and C. Barb, "An integrated experimental enviroment for distributed systems and networks," in *Proc. USENIX OSDI Conf.*, Dec. 2002, pp. 255–270.
[117] J. Touch, "Dynamic internet overlay deployment and management using the x-bone," *Comput. Networks*, pp. 117–135, July 2001.
[118] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," in *Proc. ACM CCR*, Jan. 2003, pp. 59–64.
[119] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *ACM SIGCOMM Conf.*, 2002, pp. 10–20.
[120] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," in *Proc. ACM SIGCOMM Conf.*, 2002, pp. 20–30.
[121] *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*. Washington, DC: Nat. Acad. Press, 2001.
[122] IETF Forwarding Control Element Separation Working Group Home Page. [Online]. Available: http://www.ietf.org/html.charters/forces-charter.html
[123] C. Partridge, A. Snoeren, T. Strayer, B. Schwartz, M. Condell, and I. Castineyra, "FIRE: Flexible intra-AS routing environment," in *Proc. ACM SIGCOMM Conf.*, 2000, pp. 191–203.
[124] A. Galis, B. Plattner, J. Smith, S. Denazis, E. Moeller, H. Guo, C. Klein, J. Serrat, J. Laarhuis, G. Karetsos, and C. Todd, "A flexible IP active networks architecture," in *Proc. 2nd IWAN*, H. Yasuda, Ed., 2000, pp. 1–15.
[125] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson, "Practical support for IP traceback," in *Proc. ACM SIGCOMM Conf.*, 2000, pp. 295–306.
[126] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakuontio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *Proc. ACM SIGCOMM Conf.*, 2001, pp. 3–14.
[127] J. Mitola III, "Software radios," in *Proc. IEEE National Telesystems Conf.*, May 1992.
[128] V. Bose, "Virtual Radios," Ph.D. dissertation, Mass. Inst. Technol., Cambridge, MA, 1999.
[129] F.-M. A. (Italian proverb 'La meglio e l'inimico del bene') Voltaire, Contes, 1772.

Jonathan M. Smith (S'86–M'89–SM'93–F'01) received the Ph.D. degree in computer science from Columbia University, New York, NY.

He is the Olga and Alberico Pompa Professor of Engineering and Applied Science at the University of Pennsylvania, Philadelphia, and a Professor in the Computer Information System Department. His research is centered on advanced communication and computer networking systems. He was previously at Bell Telephone Laboratories and Bellcore, Piscataway, NJ, where he focused on UNIX internals, tools, and distributed computing technology, and was a member of a technology transfer team for computer security. At the University of Pennsylvania, he has worked on advanced communications systems, such as gigabit networks, on which he has written extensively and has several U.S. patents. His current research interest is programmable network infrastructures: "Protocol Boosters" provide a methodology for using such infrastructures and "SwitchWare" is an idealized programmable infrastructure. He has consulted extensively for industry and government.

Dr. Smith is a member of ACM and Sigma Xi.

Scott M. Nettles (M'03) received the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1996 for work on high performance storage management and garbage collection for transaction systems.

He is an Assistant Professor of electrical and computer engineering at The University of Texas (UT), Austin. His research concerns broad aspects of programming languages, computer systems, and communication networks. He was previously an Assistant Professor in the Computer Information Systems Department at the University of Pennsylvania, Philadelphia, a Visting Assistant Professor in the University of Arizona, Tuscon, Computer Science Department, as well as a Member of the technical staff at Digital Equipment Corporations Western Research Lab., Palo Alto, CA. Since going to the University of Pennsylvania in 1995, his work has focused on applying programming language technologies to active networks. He joined UT in 1999, where in addition to active networking, he is working on problems concerning the interaction of wireless physical layer technologies with the link, network, and higher layers.