January 2003

Reflections on Active Networking

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-03-05.

Reflections on Active Networking

Abstract

Interactions among telecommunications networks, computers, and other peripheral devices have been of interest since the earliest distributed computing systems. A key architectural question is the location (and nature) of programmability. One perspective, that examined in this paper, is that network elements should be as programmable as possible, in order to build the most flexible distributed computing systems.

This paper presents my personal view of the history of programmable networking over the last two decades, and in the spirit of "*vox audita perit, littera scripta manet*", includes an account of how what is now called "Active Networking" came into being. It demonstrates the deep roots Active Networking has in the programming languages, networking and operating systems communities, and shows how interdisciplinary approaches can have impacts greater than the sums of their parts. Lessons are drawn both from the broader research agenda, and the specific goals pursued in the SwitchWare project. I close by speculating on possible futures for Active Networking.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-03-05.

Reflections on Active Networking

Jonathan M. Smith*
CIS Department, University of Pennsylvania
jms@cis.upenn.edu

Abstract

Interactions among telecommunications networks, computers, and other peripheral devices have been of interest since the earliest distributed computing systems. A key architectural question is the location (and nature) of programmability. One perspective, that examined in this paper, is that network elements should be as programmable as possible, in order to build the most flexible distributed computing systems.

This paper presents my personal view of the history of programmable networking over the last two decades, and in the spirit of "*vox audita perit, littera scripta manet*", includes an account of how what is now called "Active Networking" came into being. It demonstrates the deep roots Active Networking has in the programming languages, networking and operating systems communities, and shows how interdisciplinary approaches can have impacts greater than the sums of their parts. Lessons are drawn both from the broader research agenda, and the specific goals pursued in the SwitchWare project. I close by speculating on possible futures for Active Networking.

1 Introduction

Distributed computing systems and remote access to computing resources are major reasons for computer networks[Bar64, Bar02]. Thus, a central question in any network design is its effectiveness as an architecture to support distributed computing. Effectiveness can be achieved in a variety of ways, ranging from the performance criteria such as bandwidth and reliability, to more subtle measures such as scalability and the ease with which new software features can be added to the system.

The current Internet has done very well with respect to scalability, and has been very effective in absorbing new technologies which increased throughput and reliability, as well as supporting new overlaid applications. For many kinds of software, the ability to add features at the "edge" of the network is sufficient. Yet for other distributed computing models, such as multicast, which

*The work in this paper was supported by DARPA under Contracts #N66001-96-C-852 and #DABT63-95-C-0073, and the National Science Foundation under grants #ANI-9906855, #ANI98-13875 and #ANI00-82386

require network-layer participation in the distributed computation[Dee89], the Internet has been far less accommodating. This resistance to change extends to the network-layer itself, as conversion to IPv6[DH96] has been very slow.
This stems from several factors, of which I think the top three are:

1. There is a widely held belief in a rule of thumb for engineering network services called the "end-to-end argument". A succinct statement of the rule is: "The network should not implement functions that the end nodes will have to replicate". The original argument was for a careful examination of where features are to be placed in engineering a distributed system. Unfortunately the argument has been misread[1] to imply that functionality which *can* be located at the system endpoints *should* be located there. A skeptic might observe that under the latter interpretation, the Internet would be source routed, and yet it is not. Some reasons (consistent with the paper) are that routing appears to work more effectively if done in the network itself, due to: (1) fast response to changes such as failures; (2) incomplete topology information; (3) the ability to construct global routes with local decisions; and (4) information reuse (the route from Penn to the University of Cambridge is likely to overlap significantly with the route to University College London, reducing required state at intermediate gateways). Another argument for driving functionality to the edges, of course, is that it avoids the problem of changing IP.

2. The process by which a new feature is added to the IP network is a standardization process, which to be succinct operates at a political pace rather than a technological pace. With the number of parties interested in the Internet continuing to increase, this political pace slows at the same time that the technological needs are accelerating. While the IETF and IESG have made a significant effort to stay flexible, it is perhaps too much of a challenge to address a problem with all standards processes. In any case, the real limit on Internet improvements will always be the intellectual challenge of understanding complex problems and justifying their solutions, but the need for consensus further delays deployment of these solutions.

3. First with the success of NSFnet in the mid-1980s, and then later with the Internet's commercialization, the Internet has in many ways escaped the ability of computer scientists and network researchers to control its evolution. While in many ways it is attractive to have this genie out of the bottle, the cost is that experimentation has become far more difficult than it was in the early days of the Internet. The result is that any large scale experiments, other than measurement, are now done via simulation rather than with the Internet itself.

These limitations of the Internet, and a belief that a distributed system architecture may be more effective with some of its functions embedded in the network, lead to a natural research question. That is: how would one identify such features, and what sort of network architecture would be necessary to support the design of many varieties of new distributed computing systems.

[1] Interestingly, the paper notes "an incomplete version of the function provided by the communication system may be useful as a performance enhancement", foreshadowing the Protocol Boosters architecture described in Section 4.

1.1 Outline of the paper

In this paper, I begin by looking back to distributed computing systems of the 70s and 80s to frame some of the technological evolution - and research - which has gotten us to where we are today. I then discuss how some of the concerns about network architecture I have listed led to the "Store Translate and Forward" model of networking. It is the STF model which led to the current renaissance in thinking about network architecture as being driven by distributed computing rather than being engineered in isolation. Two examples of this renaissance are the design of protocol boosters and the SwitchWare project. In the material below, I mix in some 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.

2 Distributed Computing and Active Networks

Process migration, was to the best of my knowledge first implemented by David Farber's DCS system [FL70, Far73] in the mid-1970s. Over the course of the 1980s, a large number of researchers, including myself, attempted to realize process migration with varying degrees of success (see my now somewhat dated survey [Smi88] for details). The basic idea 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.

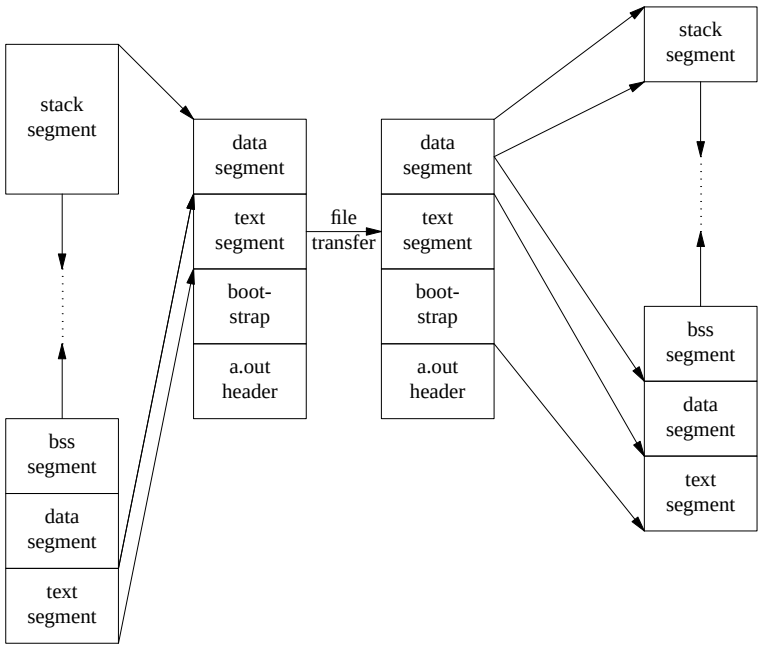

Figure 1: User-Level Process Migration using Checkpoint/Restart

At the time I wrote the survey, 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. In 1986, I implemented a system, later enhanced with help from John Ioannidis[SI89], with which a process could migrate itself. The basic idea is illustrated in Figure 1; the enhancements consisted of using NFS to save the executable image, while sending the image's name with a single UDP datagram to a server. The server then used the name to fetch the code and continue execution (this is analogous in some ways to the approach pursued in ANTS[WGT98]).

The system suffered from some fairly severe constraints, *e.g.*, that it needed a daemon, couldn't 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 towards writing distributed applications.

Based on this experience, I argued for the use of symbolic representations of state, as I believed was possible in languages such as Lisp[SGQM88]. This same notion was explored, for example, by Falcone[Fal87]. Another important discovery was that the entire program need not be moved, only parts of it[Sta86, SG90b, SG90a]; this approach is referred to as "remote evaluation" although other code-shipping approaches were being explored concurrently, such as late-binding RPC[Par92].

In a very deep sense, mobile code schemes followed remote execution and remote evaluation models, providing a general solution to "process migration" once appropriate language technologies became available. The capsule approach to active networking is simply an application of these mobile code techniques and technologies to the domain of networking.

3 The broadband Internet

In the early 1990s[Sta90], the Internet was coming into its own as a network infrastructure. A large group of researchers was exploring methods of increasing network throughputs, by a factor of 100, using a variety of technologies, such as SONET[Bab90, SON91], HIPPI[Gli92, HIP94, JD93] and ATM[GHM+91, BDH+92].

The really interesting possibilities here were those of building distributed computing infrastructures extended to wide areas. Low bandwidth in the core 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 analyses and weather modelling, that were unachievable without access to remote computational resources and data.

Research in the AURORA gigabit testbed[CDF+92, Cla93], in which I was involved, was centered around Asynchronous Transfer Mode (ATM) technology. While ATM signalling never quite matured, ATM link layers led to the broadband Internet[Sin02], both in providing an infrastructure for high-speed IP overlays, and then later evolving into the methodology for building high performance IP switches[NML98]. Once ATM signalling was (wisely) replaced in architectures such as MPLS[Li99] which provided virtual paths without heavyweight signalling, the basic advantages of the technology became available to Internet users, and its impact is still felt as it is used in Digital Subscriber Lines as well as inside switch backplanes.

The Internet[Cla88] solves an interesting problem, that of providing a universal networking infrastructure[Tan88]. It addresses the problem 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 sending device formats an IP packet and encapsulates it in one or more frames of an attached network type, such as Ethernet. 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 signalling 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 signalling protocols. This later generated an energetic line of research[HR98, vdML98] at the University of Cambridge, which developed virtual signalling stacks to allow customization. In many ways, the Cambridge work, which they describe as "Open Signalling", is analogous to an active networking approach limited to the network control plane.

In discussions during a visit to Penn in the early 1990s, Dave Sincoskie observed that the telephone network achieved interoperability with a circuit model based on a 20mA 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]. He posed to me the problem of pulling the best features out of these two examples, to take the lessons learned, and apply them to a new architecture in which service introduction could be accelerated. Dave presented this problem as "interservice networking", and presented the notion of tying together services such as voice, FAX and IP. While I spent the great majority of my time working on gigabit networking, this question nagged at me. Finally, during a Fall break in Maine in 1993, in what I thought to be a completely original idea (only to be pointed to Softnet several years later by Bryan Lyles; see discussion below in Section 5) I wrote a short abstract of the idea and e-mailed it to Dave Sincoskie and Dave Farber (my colleague and research mentor), which I include as an Appendix.

The more I considered this problem, the more I became interested in Store-Translate-and-Forward as a networking model, as it provided a strong "Computer Science" flavor to networking. Elementary computability teaches us that translators are computers, computers are translators, and so on. Thus, STF became my focus, and I began to think about how to make it happen. At that point I did not know quite how to do STF, but it was clearly *possible* simply by inserting a programmable general purpose computer into the network, so research was warranted.

In addition to the basic architectural notion, my 1993 e-mail (with authentic grammatical and typographic errors) also suggested a few applications, which I include as a second Appendix, both for completeness, and because some remain interesting, almost a decade later.

So inserting translators in the network was 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. The next section, on "Protocol Boosters", explains how this came about.

[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

4 Protocol Boosters

In 1994, David Feldmeier and I discussed collaborating on a DARPA proposal submission. Bellcore had begun seeking external research funding, and David had some interesting ideas on protocol architectures. I suggested building a Desktop-Area-Network (DAN) based on my STF idea. We put together a response to DARPA on this idea, for which we used Dave's name (which I still like!) of "Protocol Boosters". The idea there, to recap, is that we can *dynamically construct* protocols using as-needed element insertion and deletion, to respond to network dynamics. The paradigm was to construct protocols from elements optimistically; that is, elements were inserted into protocols on-demand, as conditions were encountered that required the protocol element. This embedded my ideas about how to build loadable modules which performed translation tasks.

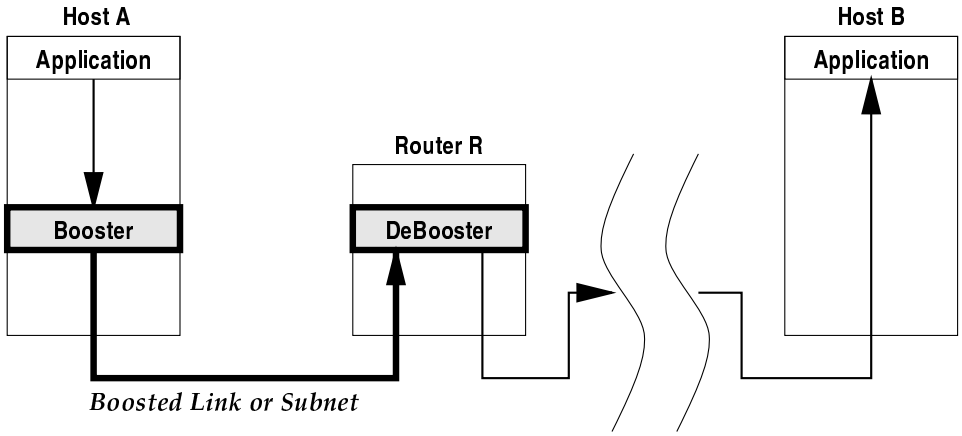

Figure 2: Use of a protocol booster on a network link

Our proposal was a significant break from convention in that we were willing to alter the network fabric, in conflict with (at least the misreading of) the "end-to-end argument" discussed above in Section 1. A major goal was the ability to accelerate network evolution; we predicted that we could enable 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. We felt that a marketplace that was more likely to develop was one of proprietary protocols developed by protocol vendors, to be sold or deployed in their own networks to achieve a competitive advantage. However, in this research effort, we focused mainly on whether protocols could be constructed using "as-needed" techniques, and if so, what form these protocols would take.

4.1 Boosted Protocols

An example that we used extensively, both to explain and to experiment with Protocol Boosters, was Forward Error Correction, or FEC. FEC is a technique which 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.

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 looking into the future remains an unsolved problem, so we needed mechanism to adapt the protocols "on-the-fly", but this approach appeared to be (and later proved to be) quite attractive, since its benefits could be easily quantified and therefore demonstrated experimentally.

4.2 Infrastructures and Experimental Results

The two most important influences on our initial attempts at implementation were Hutchinson and Peterson's[HP91] *x*-Kernel system and Dennis Ritchie's STREAMS[Rit84] architecture. Ritchie's system provided an elegant architecture 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 we had envisioned for Protocol Boosters, but we felt that the programming model was (no doubt on purpose) too restrictive for what we had in mind for boosters. 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 made heavy use of compiler techniques to connect a *protocol graph* of protocol components together into a system. This style of protocol composition was more to our liking, but after investigating, it appeared that the existing *x*-Kernel tools did not permit dynamic reconfiguration of the protocol graph.

We chose to build a system combining the dynamics we desired in protocol construction with the generality of the *x*-Kernel's composition architecture.

Our first attempt[MCS97] 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 BSDLite stack is shown in Figure 3; further modifications (basically a small multithreading system) were made to handle more complex boosters, as shown in Figure 4.

The small system we had built showed that a flexible and dynamically modifiable system could be built, thus we developed a far more mature kernel infrastructure[MMR98, FMS+98] 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.

Given that protocol boosters were used for many bit-intensive tasks (FEC, compression and encryption) their implementation in software presented performance challenges. We decided to

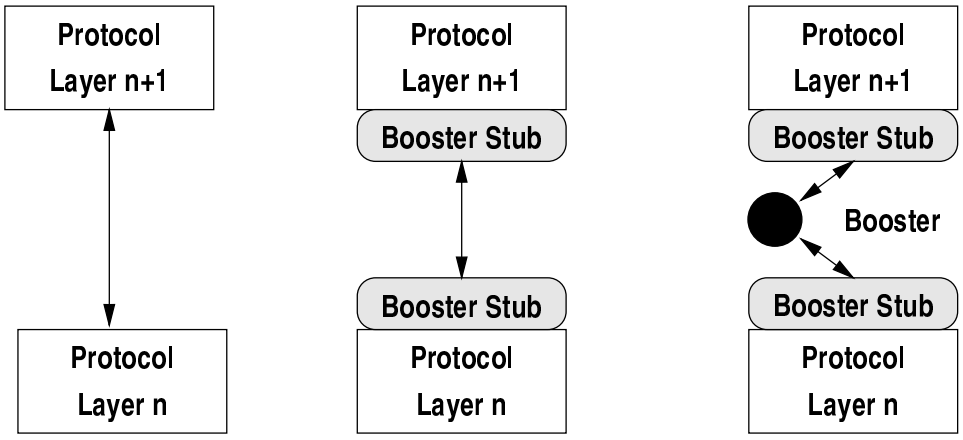

Figure 3: Inserting a booster into a protocol stack

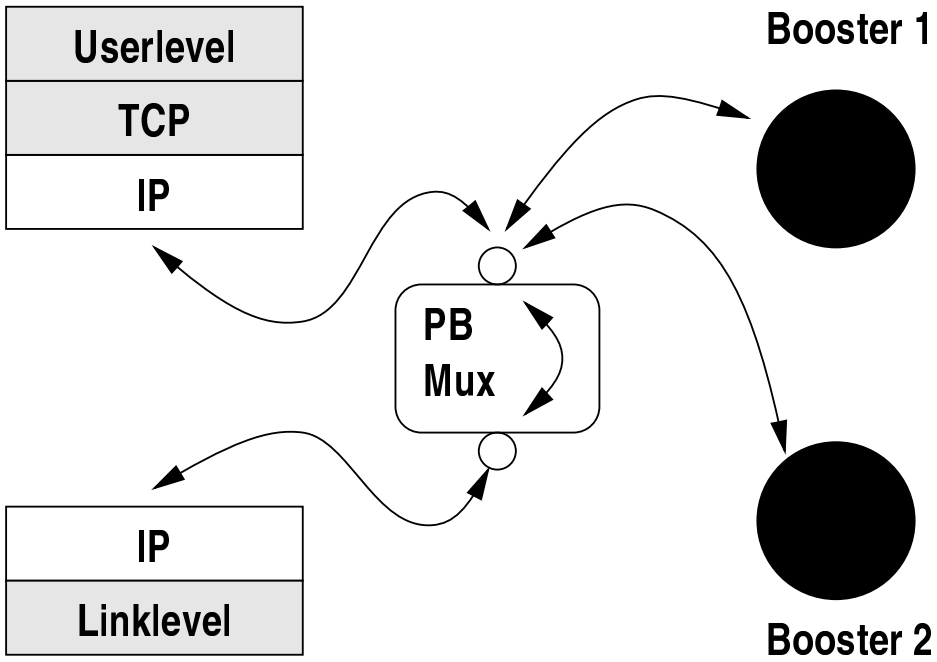

Figure 4: Multiplexing of boosters in the 4.4 BSDlite IP Stack

take the basic boosters ideas and to build hardware support[HS97], 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 Figure 5.

The P4 was used standalone to demonstrate that boosters could be run at OC-3c line rates of

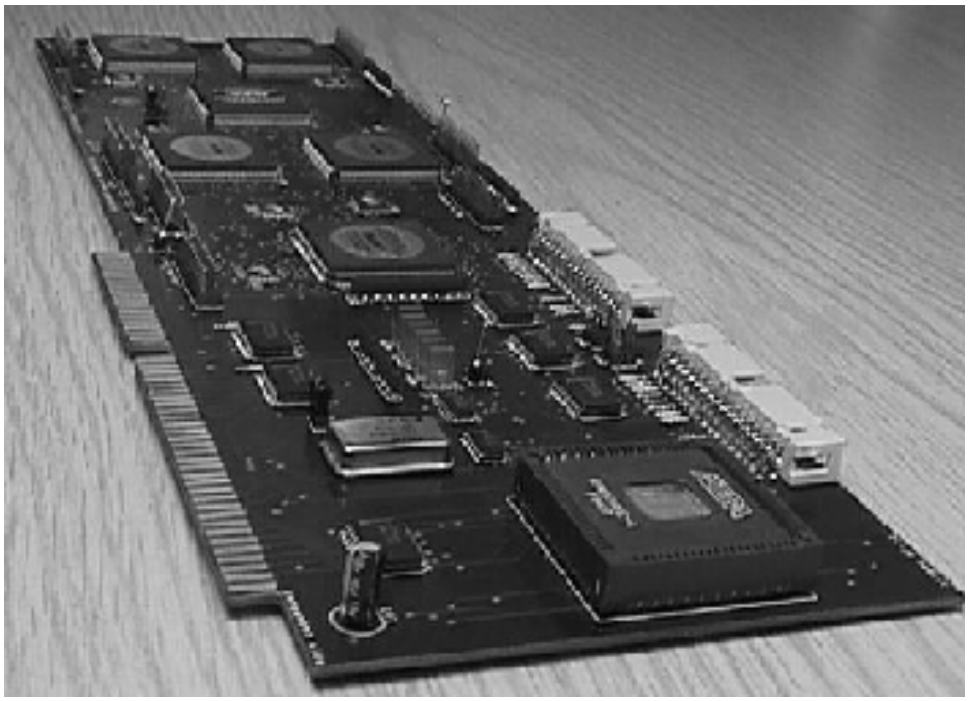

Figure 5: The Programmable Protocol Processing Pipeline

155 Mbps, and 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[Had99] dissertation demonstrated that an FEC booster could automatically detect the need for insertion using failed AAL-5 CRCs, insert itself, and enhance the performance of a set of TCP/IP/ATM benchmarks, thus demonstrating the power of the protocol architecture.

4.3 From Protocol Boosters to DARPA's Active Networking program

One thing that was becoming clear was the need for a general purpose infrastructure with which new protocols could be developed, something we had initially decided was not a main focus of Protocol Boosters. Fortunately, the flavor of Store-Translate-and-Forward in Protocol Boosters had attracted the attention of others.

Several years after the fact, during a discussion with Gary Minden, the DARPA program manager responsible for funding the "Protocol Boosters" proposal, I discovered that DARPA had been internally discussing network/computer integration (NCI), and that Protocol Boosters was funded as it was a concrete proposal of a way 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, the one which led to the DARPA Active Networks program.

5 Active Networks

At SIGCOMM 1995 in Boston, I explained Protocol Boosters and my ideas on Store-Translate-and-Forward to David Tennenhouse, with whom I had worked in the AURORA Gigabit Testbed [Cla93] which connected Penn, Bellcore, IBM Research and MIT in the Northeastern U.S.. Tennenhouse told me about his own plans for a programmable network infrastructure, which he called "Active Networks". Seeing that we shared a vision, we agreed to work together to make these visions a concrete reality.

Tennenhouse had been serving on the ISAT team that developed the Active Networking Program. We agreed to work on a joint publication on the ideas and vision to try to set the agenda for the research. In the meantime, Gary Minden at DARPA had successfully moved the results of the ISAT study towards a "Broad Agency Announcement" or BAA, which is DARPA's means for soliciting research proposals, and this BAA would be ready later in the Fall.

A sizable group of researchers[Yd96, HMPP96, PJ96, SJS+00, TW96, SFG+96] were involved with putting together a collaborative set of responses to the BAA. Teleconferencing was used to put together 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 six areas: (1) Enabling Technologies, (2) Platform Development, (3) Programming Models, (4) Middleware Services and Applications, (5) Active Controls and Algorithms, and (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. The research agenda was broadly similar to the one that David Tennenhouse and I had discussed in a walk along the Charles River in the Fall of 1995; this research agenda was later captured in a paper by Tennenhouse and Wetherall[TW96].

In any case, in preparation for writing our response to BAA 96-06, "Accelerating Network Evolution with a Software Switch for Active Networks", we began to study what was known in order to distinguish the research from what had been done. In a discussion with Bryan Lyles, then of Xerox PARC in which I had described my STF model, Lyles pointed me to some beautiful early work[3] done by Zander and Forchheimer of the University of Linkoping on a system called "SOFTNET". The SOFTNET [ZF80, ZF83] 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 "Active Networking".

Interestingly, at around the same time, the progenitor of what was originally called the "Capsules" [TSS+97] model and is now more commonly called the "Active Packet" model[HMWN02]

[3]Leading of course to that phenomenon every researcher is familiar with, of discovering your "new idea" in an ancient reference!

was being developed by David Wall[Wal82]. In this paper, Wall outlined a new approach to networking. I quote from the 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 some ideas such as process migration were captured earlier still by Farber's DCS System[FL70, Far73], DCS was focused on distributed computing rather than a rethinking of how to *program* network functions.

While neither the SOFTNET nor the Active Message system captured the entire Active Networks agenda, they had all the foundational ideas. The advances then, would be from new technologies that had arisen since, in particular new programming language[MTH90, GJS96] and security technologies[BFL96] that could provide desirable sets of tradeoffs amongst security, programmability, usability and performance.

The next section covers the results of our research effort, which we entitled "Accelerating Network Evolution with a Software Switch for Active Networks". We had initially called the project "SoftSwitch", but after some concerns David Farber raised that this name might be trademarked, we renamed the project "SwitchWare"[4].

6 SwitchWare

We believed in 1995[SFG+96] that the research challenges included implementation of security (e.g., defining sets of allowable programs) and resource management policies (to capture notions such as time requirements for SwitchWare switched traffic with QoS requirements), which must be embedded in the SwitchWare switch's Operating System/run-time environment. The question was how to achieve them?

We were confident that programming language technologies such as strong typing[MTH90] and garbage collection[NOG93] were the "secret sauce" with which we could achieve desirable tradeoffs, but we were not completely sure about how to proceed. We could build upon the Protocol Boosters work described earlier, which was still underway, or we could embark upon an entirely new path. Since Protocol Boosters had been focused on protocol design, that project was rather agnostic with respect to the programming environment (in fact, multiple Protocol Boosters environments were built, for FreeBSD, Linux and Altera field-programmable logic devices), so we chose the "take a new path" approach.

We were fortunate to have a team of people almost tailor-made to pursue such an aggressive research agenda. In particular, Scott Nettles, a recent Ph.D. from CMU, had joined the Penn faculty and had a unique blend of programming language and operating systems technology expertise. This allowed the team to take advantage of the new programming methodologies I had alluded to earlier, and to leverage the work of the ML community, which was increasingly interested in networking applications [Bia94].

Our major focus was on developing a programming environment with which new network functions could be dynamically inserted and deleted. We wanted to achieve high performance, and felt that a high performance environment could be built if we specialized a programming environ-

[4] Which turned out to be trademarked as well!

ment for networking. The notion was simple: the compiler could be relied upon to make many safety and security decisions *statically*, at compile time, rather than dynamically at execution time. Our belief was that if we created a "software switch" model, improvements in computer hardware would be easy to exploit, and further, that the evolution of network algorithms and services could be accelerated.

We developed two programming environments in the SwitchWare project, the ALIEN[ASNS97, Ale98] active loader and the Packet Language for Active Networks (PLAN)[HMA+99] (PLAN served as the inspiration for the later SNAP[MHN01, Moo02] system). The architecture of the final system is shown in Figure 6.

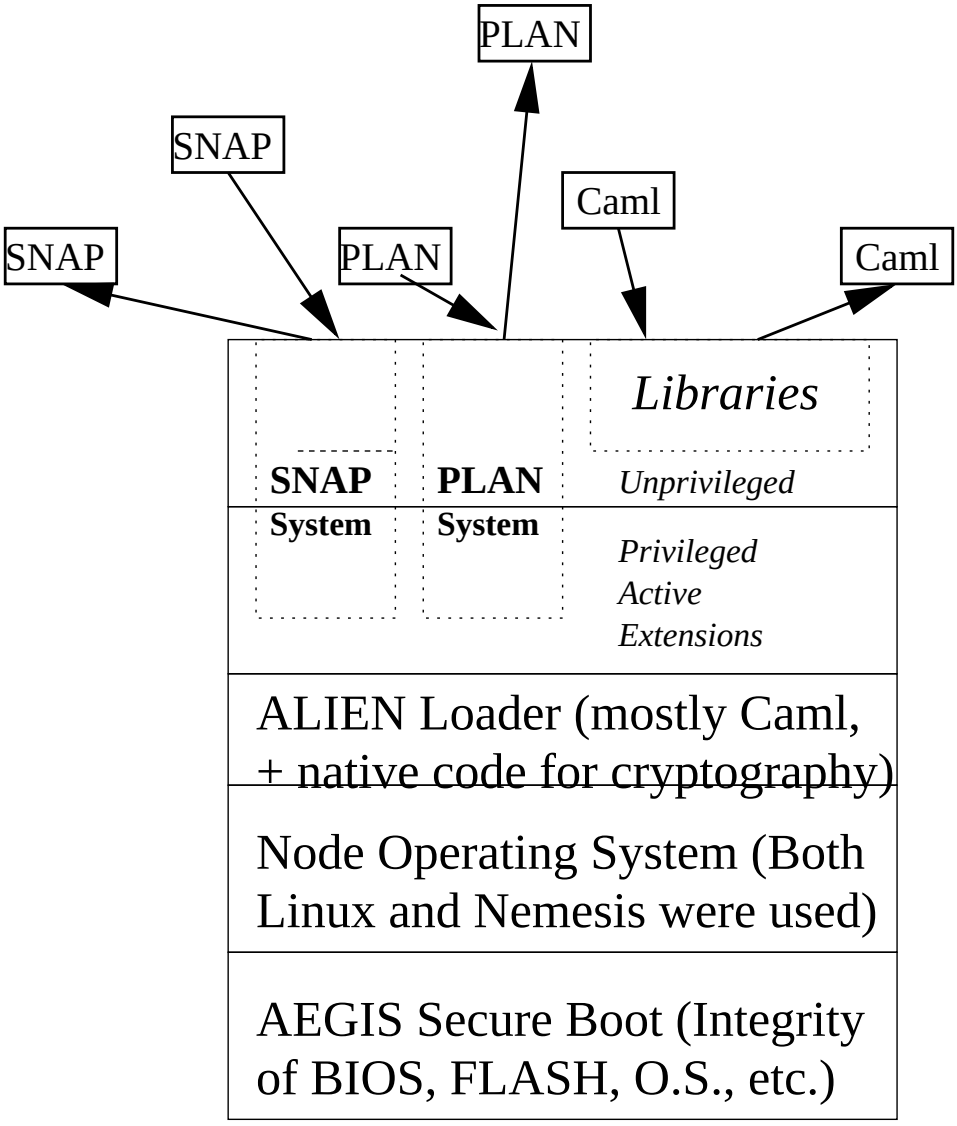

Figure 6: The SwitchWare system architecture

Our original goal had been to build an environment suitable for a network element, such as a

bridge or a router. The ALIEN system, first prototyped in the active bridge[ASNS97], showed that a complete bridge could be constructed "on-the-fly" from modules that added buffered repeating, trivial LAN extension, self-learning and spanning tree functionalities. Then, as we felt that a major issue would be the robustness of the system with failed software, we built two spanning tree modules, one with an error in the module's implementation, and fell back to the correctly functioning module after a logical self-check failed.

While we used a TFTP implementation to implement the active bridge, we extended the active loader to check cryptographic credentials associated with modules. This supported remote use of the Caml[Ler95] "module thinning" approach we used to control privilege in ALIEN. After adding a secure bootstrap (AEGIS[AFS97]) to prevent subversion of our privilege enforcement from below[5], we called the resulting system SANE[AAKS98a], for "Secure Active Network Environment". 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 the loader to be extremely "thin", as hinted at in the illustration.

While we initially used Linux as a development platform, it became clear to us that in the role of a network element operating system, even if booted securely, Linux had severe limitations. In particular, it did not share the careful attention paid to security issues that we had focused on ALIEN, and the general distributions had no support for resource guarantees, the only way of addressing the threat of denial-of-service attacks[Nee94]. We addressed these issues with two thrusts. First, we absorbed development of a secure operating system based on capabilities, EROS[SSF99], into the project. At one point we speculated that we might move all development to EROS, but its maturity and infrastructure in 1997 were insufficient to allow this at the time, so we kept Linux as the major development platform. The second thrust was resource guarantees for resources such as memory, bandwidth and computation time. We addressed the denial-of-service attacks in one fashion in our Secure Quality of Service Handling[AAK+00] (SQoSH) architecture in one fashion, and in another fashion entirely in the Resource Controlled Active Networking Environment[AMK+01] (RCANE).

SQoSH used a new operating system called Piglet[MS98a, MS98b, Mui01], which had originally been intended for smart line cards, but turned out to be ideally suited for multiprocessors (and continues to look promising for both smart line cards and for network processors). 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. 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.

The RCANE project was a collaboration with the University of Cambridge, in the person of Paul Menage. Paul had spent a summer internship at Penn, and quickly grasped the essentials of the SwitchWare approach and architecture. He had significant experience with a new multimedia operating system, Nemesis[LMB+96], developed at the Cambridge Computer Laboratory. Paul examined the resource issues, including garbage collection and CPU resources and devel-

[5]This focus on top-to-bottom integrity led to the discovery of new classes of security threats for programmable hardware[HUS99]

oped a complete resource management architecture[Men00] which addressed QoS management, QoS crosstalk and was completely integrated with SwitchWare[AMK+01], including support for PLAN.

The SwitchWare node architecture addressed all of the fundamental problems of security[AAH+98, AAKS98a, AAKS98b, AAKS99]: 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.

While conscious of these achievements for the network element view of active networking, we were sensitive to the assertion that the "capsules" model was more aggressive than our approach, and were intellectually curious about active packet *versus* active extension tradeoffs. We believed that it would be easier to build an active packet system from an active extension system (such as ALIEN) than it would be to build an active extension system using active packets (particularly in a system such as ANTS, which employed soft-state code caches). In addition, there were some very deep questions regarding the separation of concerns between what was necessary for safety, security and performance for active packets as opposed to active extensions. For example, if one accepted the model that users would deploy changes via active packets and administrators (*e.g.*, ISPs) would control offered services via active extensions, then the active packet language would be rather restrictive, but would need the capability to reference arbitrary services.

PLAN[HKM+98] was developed by Nettles, Gunter and their doctoral students to investigate the idea of using a *domain-specific language* for active networking. This would have two goals. First would be the ability to restrict the language to a set of simple operations that almost all nodes would willingly execute, in the style of the ICMP ECHO packet ("`ping`"). Second would be a vehicle suitable for formal specification.

To achieve the first goal, the language was designed as a "glue" language which could be used to compose operations provided by active extensions. The active extension system (*i.e.*, ALIEN) was then charged with managing authentication and authorization. This allocation of roles is similar to that used in the UNIX model of an unprivileged command interpreter used with commands and a file system which control privilege. The PLAN architecture has proven sufficiently powerful that it has been used to build an active internetwork, as well as applications such as flow-based adaptive routing (FBAR).

Towards the second goal, a formal specification of PLAN was produced by Kakkar, *et al.* in 1999[KHMG99]. PLAN was used in formal methods work at SRI and elsewhere in the Active Networks work due to its desirable resource-boundedness.

The SwitchWare system as a whole was used to develop user applications; Bellcore (now Telcordia Technologies) designed and implemented a network publish/subscribe application using PLAN and the SwitchWare software which showed the power of the technology in dynamically constructing network services.

Figure 7 illustrates my view of what was achieved in the active networking design (sub)space of performance versus flexibility tradeoffs. The P4 alone, while operational at 155 Mbits/sec and workable to 622+ Mbits/sec[Had99] with different component choices, is perhaps the least flexible of the systems, as in standalone configurations it can only accommodate "programs" which

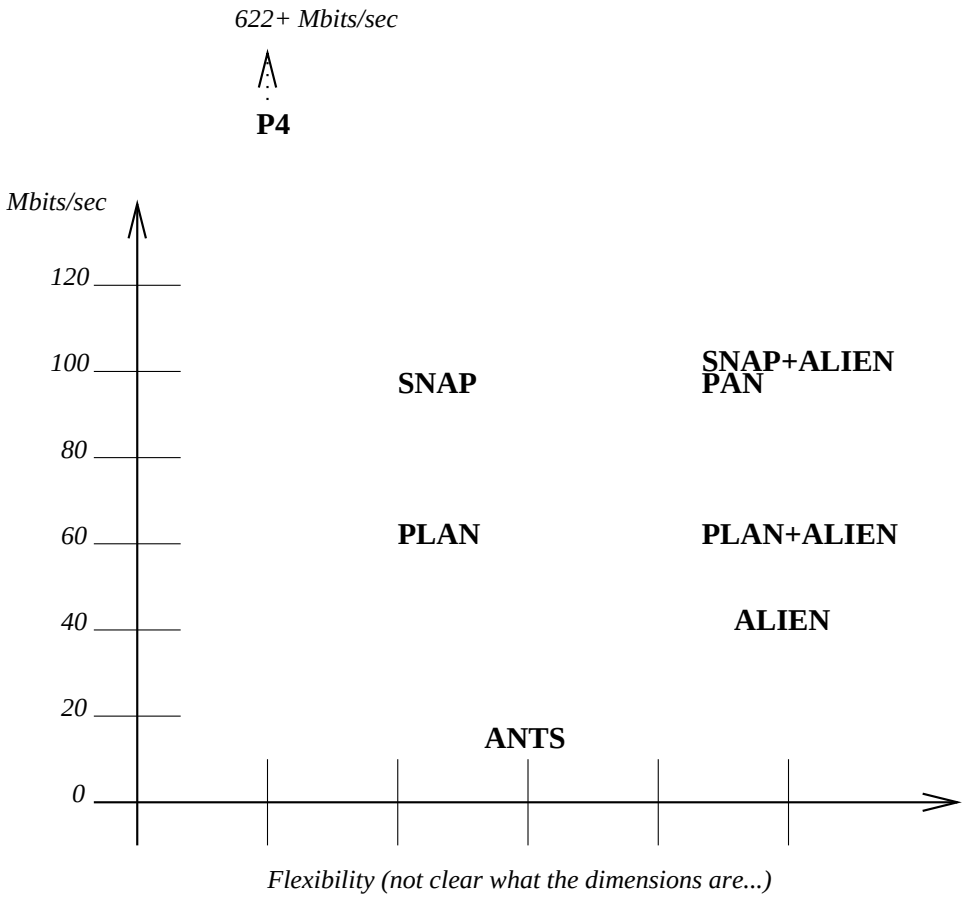

Figure 7: Flexibility versus Measured Performance Tradeoffs

can be expressed within the constraints of a field-programmable logic device[6]. I argue that ANTS is less flexible than ALIEN because ANTS could be implemented in ALIEN, while the converse is not true. I argue that PLAN and SNAP are less flexible than ANTS standalone (that is, ignoring the capability for active extensions), while in concert with active extensions (indicated by PLAN+ALIEN and SNAP+ALIEN[7], respectively) they absorb all the flexibility inherent in the active loader architecture. 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

[6]Two legitimate counterarguments could be made. First, as a modification of the basic hardware elements, the gate array is an *extremely* flexible component. Second, when used in combination with other technology such as ALIEN, as the P4 was in its final instantiation[Had99], the architect had access to all levels of the protocol architecture, down to the hardware

[7]While SNAP+ALIEN has not been implemented, the integration of the SNAP virtual machine with the active extension support of ALIEN is obvious given the integration of the PLAN virtual machine with ALIEN

packet level, and with the addition of some operating 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 lead me to conclude 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.

The combination of SNAP and ALIEN yields simultaneously the most flexible, highest performance and most secure active networking system, based on performance reported for SNAP.

A natural question is why the SwitchWare software did not gain wider acceptance, given its technical merits. It is my belief that this was a function of our attempt to solve "the whole problem" and to use the best technologies we could find or create. For example, the considerable strengths of Caml as a programming language were overwhelmed by the popularity of Java and the resulting population conversant with Java-based approaches such as that used by ANTS. Given a choice between a Caml-based system and a Java-based system such as ANTS, programmers judged ANTS "good enough" and succumbed to intellectual inertia in not learning Caml. We should have remembered that "*le mieux est l'ennemi du bien*"[Vol72].

7 Conclusions and towards the future

The original DARPA Active Networks program is now drawing to a close, although some work remains underway. As the program was begun at the dawn of the Internet's large-scale commercialization, considerable skepticism from the research community arose, some of it legitimate, regarding performance and security. My belief is that the security issues are largely addressed, as discussed in Section 6.

The performance is adequate for the network edge, as illustrated in Figure 8, which illustrates the number of programmed instructions (P) that can be executed by a general purpose processor without delaying the data transmission. Specifically, almost all access networks have bandwidths less than 100 Mbits/sec, a speed at which many of the systems described in this paper operate at, or near. Supporting evidence for this claim are the domain-specific "middleboxes" appearing there such as firewalls, NATs and IDSes.

Additionally, new *network processors*[Int, IBM, Age] 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.

Worldwide efforts in active and programmable networking are underway, particularly in Europe (see, *e.g.* the "Future Active IP Network" project[GPS+00] or the ALPINE project[ALP]). 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"[SWKA00, SPS+01] are penetrating products.

The IETF's Forwarding and Control Element Separation (ForCES) working group[For] 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[PSS+00]. The wireless industry is also

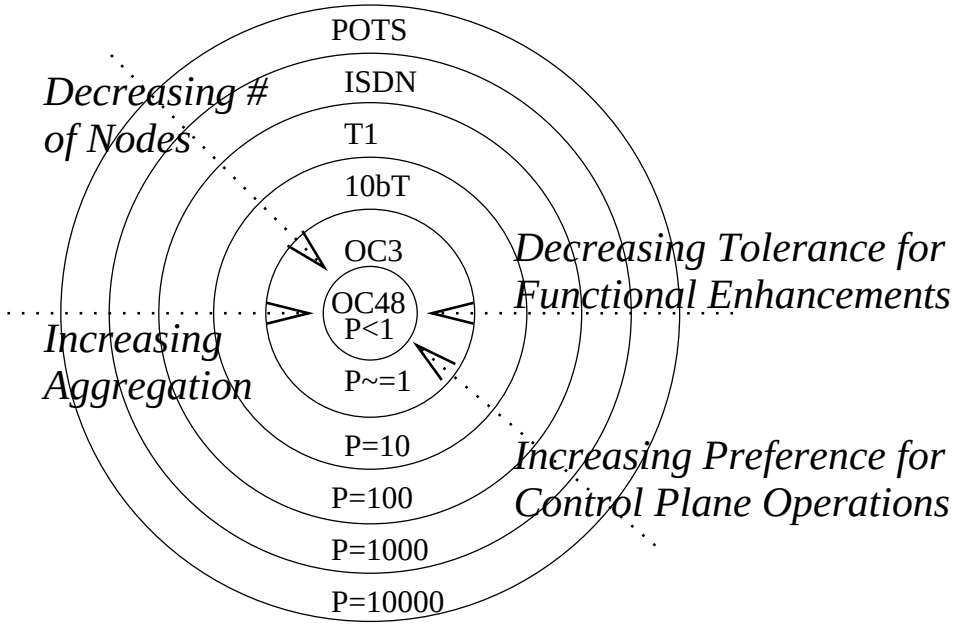

Figure 8: Computation versus Network Performance Tradeoffs

absorbing active network approaches, ranging from mobile telephony industry's use of mobile code to the software definition of radio[III92, Bos99].

Nonetheless, the need for network evolution remains, and in many ways the needs of distributed computing are still not well met by the exclusion of computing APIs from the network infrastructure. The uptake on active networking has been slow, and its short-term impact has not been sufficient. The active networks work may have been wrongheaded, it may have been premature, or it may have been pursued the wrong way. While there is an element of the second, I believe that the root cause the third of these, as I see the current focus on peer-to-peer systems as a way to introduce new services using an overlay, in the style of the World Wide Web. If peer-to-peer systems become sufficiently successful, they may stimulate selective reshaping of the network layer from above, at the same time that software radio and network processors reshape it from below.

In summary, Active Networks offer an opportunity to build a truly flexible distributed computing infrastructure, where the programmer was in command of all aspects of their distributed computing model. The market is delivering network embedded programmability, albeit in an ad-hoc and purpose-built fashion. For example, 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. 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.

8 Acknowledgements

More than anyone else, I would like to thank Dave Sincoskie, whose challenging questions stimulated "Store-Translate-and-Forward". David Feldmeier was both a collaborator and a student, and great source of ideas - Protocol Boosters was our first stab at getting this right, and we continued to work together on Active Networking in the SwitchWare project. Bryan Lyles pointed out the excellent work done by Zander and Forchheimer in the Softnet project, and Bernhard Plattner pointed out the work done by Wall in his analysis of messages as active agents.

Scott 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. Bill Arbaugh and Angelos Keromytis figured out how to secure the prototype system as it evolved into Alexander's ALIEN system. Ilija Hadzic designed and implemented the P4, showing that programmable infrastructures could run *really* fast, and in some ways foresaw network processors. Steve Muir and Jonathan Shapiro designed novel alternative operating systems architectures (Piglet and EROS, respectively) so that we really understood the space.

My collaborators at Penn in the SwitchWare project, David Farber, Carl Gunter and Scott Nettles stimulated many fascinating approaches, some different that what I had initially envisioned. Scott and his students Michael Hicks and Jonathan Moore have provided the most complete understanding of active packet systems that we now have. Bill Marcus, Mark Segal and Tony Bogovic were great collaborators at Bellcore.

Gary 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. David Tennenhouse, first at MIT and then at DARPA, championed the DARPA Active Networks program into existence. The work of Tennenhouse, John Guttag and David Wetherall resulted in the most widely used Active Networking system, ANTS.

I would also like to thank the many people who have reviewed, and improved, earlier versions of this paper with their comments on both history and my description of it. These include Dave Sincoskie, David Feldmeier, Scott Alexander, Bernie Plattner, Craig Partridge, Ilija Hadzic, Gary Minden, Angelos Keromytis, Dave Farber, Sushil da Silva, Jon Crowcroft, Michael Hicks, Spyros Denazis, Scott Nettles, Bill Marcus and Bob Braden. Any remaining inaccuracies are my fault alone.

Note to reviewers: The bibliography is extensive and perhaps overly so - the references amount to about a third of the paper length. However, given the nature of this paper and its goal of "connecting the dots" amongst many research threads, I believe it is appropriate. Constructive guidance gladly accepted!

References

[AAH+98] 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 Magazine, special issue on Active and Programmable Networks*, May/June 1998.

[AAK+00] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, S. Muir, and J. M. Smith. Secure Quality of Service Handling (SQoSH). *IEEE Communications*, 38(4):106–112, April 2000.

[AAKS98a] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. A Secure Active Network Environment Architecture: Realization in SwitchWare. *IEEE Network Magazine, special issue on Active and Programmable Networks*, May/June 1998.

[AAKS98b] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith. Safety and Security of Programmable Network Infrastructures. *IEEE Communications Magazine*, 36(10):84 – 92, 1998.

[AAKS99] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. Security in active networks. In *Secure Internet Programming*, Lecture Notes in Computer Science, pages 433–451. Springer-Verlag Inc., New York, NY, USA, 1999.

[AFS97] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. A Secure and Reliable Bootstrap Architecture. In *IEEE Security and Privacy Conference*, pages 65–71, May 1997.

[Age] Agere Network Processors. `http://www.agere.com/ enterprise_metro_access/ network_processors.html.`

[Ale98] D. S. Alexander. *ALIEN: A Generalized Computing Model of Active Networks*. PhD thesis, University of Pennsylvania, September 1998.

[ALP] Application Level Programmable Inter-Network Environment Project Web Page. http://www.cs.ucl.ack.uk/alpine/.

[AMK+01] D.S. Alexander, P.B. Menage, A.D. Keromytis, W.A. Arbaugh, K.G. Anagnostakis, and J.M. Smith. The Price of Safety in an Active Network. *Journal of Communications (JCN), special issue on programmable switches and routers*, 3(1):4–18, March 2001.

[ASNS97] D. S. Alexander, M. Shaw, S. Nettles, and J. Smith. Active Bridging. In *Proc. 1997 ACM SIGCOMM Conference*, 1997.

[Bab90] J. Babcock. SONET: A Practical Perspective. *Business Communication Review*, 20(9):59–63, September 1990.

[Bar64] Paul Baran. On distributed communication networks. *IEEE Transactions on Communication Systems*, CS-12:1–9, March 1964.

[Bar02] Paul Baran. The Beginnings of Packet Switching: Some Underlying Concepts. *IEEE Communications Magazine*, 40(7):42–48, July 2002.

[BDH+92] 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 Symposium*, October 1992.

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[Bia94] E. Biagioni. A structured TCP in Standard ML. *Proceedings, 1994 SIGCOMM Conference*, pages 36–45, 1994.

[Bos99] Vanu Bose. *Virtual Radios*. PhD thesis, MIT, 1999.

[CDF+92] D.D. Clark, B.S. Davie, D.J. Farber, I.S. Gopal, B.K. Kadaba, W.D. Sincoskie, J.M. Smith, and D.L. Tennenhouse. An overview of the aurora gigabit testbed. In *Proceedings of the IEEE Infocom Conference*, 1992.

[Cla88] D. D. Clark. The Design Philosophy of the DARPA Internet protocols. *ACM Computer Communications Review*, 18(4):106–114, August 1988.

[Cla93] D. D. Clark,*et al.* The aurora gigabit testbed. *Computer Networks and ISDN Systems*, 25(6):599–621, January 1993.

[Dee89] S. E. Deering. Host extensions for IP multicasting. Internet RFC 1112, 1989.

[DH96] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. Internet RFC 1883, January 1996.

[Fal87] Joseph R. Falcone. A Programmable Interface Language for Heterogeneous Distributed Systems. *ACM Transactions on Computer Systems*, 5(4):330–351, November 1987.

[Far73] David J. Farber. The Distributed Computing System. In *Proceedings, 1973 COMPCOM*, 1973.

[FL70] David J. Farber and K. Larson. The architecture of a distributed computer system – An informal description. Technical Report 11, Information and Computer Science, University of California, Irvine, 1970.

[FMS+98] D. C. Feldmeier, A. J McAuley, J. M. Smith, D. S. Bakin, W. S. Marcus, and T. M. Raleigh. Protocol Boosters. *IEEE Journal on Selected Areas in Communications (Special Issue on Protocol Architectures for 21st Century Applications*, 16(3):437–444, April 1998.

[For] IETF Forwarding Control Element Separation Working Group Home Page. `http://www.ietf.org/html.charters/forces-charter.html`.

[GHM+91] J. Giacopelli, J. Hickey, W. Marcus, W. D. Sincoskie, and M. Littlewood. Sunshine: A high-performance self-routing broadband packet switch architecture. *IEEE Journal on Selected Areas in Communications*, 9(8):1289–1298, October 1991.

[GJS96] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison Wesley, 1996.

[Gli92] *Gigabit Rate Transmit Receive Chip Set Technical Data*. Hewlett-Packard, 1992.

[GPS+00] 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 H. Yasuda, editor, *Proceedings, 2nd IWAN*, number 1942 in LNCS, pages 1–15. Springer, 2000.

[Had99] I. Hadžić. *Applying Reconfigurable Computing to Reconfigurable Networks*. PhD thesis, University of Pennsylvania, September 1999.

[HIP94] *HP9000 Series 700 HIPPI Interface HP J2069A*. Hewlett-Packard, 1994.

[HKM+98] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings, International Conference on Functional Programming*, 1998.

[HMA+99] Michael Hicks, Jonathan T. Moore, D. Scott Alexander, Carl A. Gunter, and Scott Nettles. PLANet: An active internetwork. In *Proceedings of the Eighteenth IEEE Computer and Communication Society INFOCOM Conference*, pages 1124–1133. IEEE, 1999.

[HMPP96] J. Hartman, U. Manber, L. Peterson, and T. Proebsting. Liquid software: A new paradigm for networked systems. Technical Report 96-11, University of Arizona, June 1996.

[HMWN02] Michael Hicks, Jonathan T. Moore, David Wetherall, and Scott Nettles. Experiences with capsule-based active networking. In *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE)*. IEEE, May 2002.

[HP91] N. C. Hutchinson and L. L. Peterson. The *x*-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[HR98] D. A. Halls and S. G. Rooney. Controlling the Tempest: Adaptive Management in Advanced ATM Control Architectures. *IEEE Journal on Selected Areas in Communication (Special Issue on Protocol Architectures for 21st Century Applications)*, 16(3), April 1998.

[HS97] Ilija Hadžić and Jonathan M. Smith. P4: A platform for FPGA implementation of Protocol Boosters. In *Field Programmable Logic 1997*, number 1304 in Lecture Notes in Computer Science, pages 438–447. Springer-Verlag, 1997.

[HUS99] I. Hadžić, S. Udani, and J. M. Smith. FPGA Viruses. In *Proceedings of 9th International Workshop on Field-Programmable Logic and Applications, FPL'99*, LNCS. Springer, August 1999.

[IBM] IBM PowerNP Network Processors. `http://www-3.ibm.com/ chips/ products/ wired/ products/ network_processors.html.`

[III92] Joseph Mitola III. Software Radios. In *Proceedings, IEEE National Telesystems Conference*. IEEE, May 1992.

[Int] Intel IXP Architecture Network Processors. `http://www.intel.com/ design/ network/ products/ npfamily/`.

[JD93] Wally St. John and David DuBois. HiPPI-SONET Gateway. In *CASA Gigabit Testbed Annual Report*, pages 47–52, 1993.

[KHMG99] Pankaj Kakkar, Michael Hicks, Jonathan T. Moore, and Carl A. Gunter. Specifying the PLAN network programming language. *Electronic Notes in Theoretical Computer Science*, September 1999.

[Ler95] Xavier Leroy. *The Caml Special Light System (Release 1.10)*. INRIA, France, November 1995.

[Li99] T. Li. MPLS and the Evolving Internet Architecture. *IEEE Communications Magazine*, 37(12):38–41, December 1999.

[LMB+96] 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 Journal on Selected Areas in Communications (JSAC)*, 14(7):1280–1297, September 1996.

[MCS97] A. Mallet, J. D. Chung, and J. M. Smith. Operating systems support for protocol boosters. In *Proceedings, HIPPARCH Workshop*, June 1997.

[Men00] Paul B. Menage. *Resource Control of Untrusted Code in an Open Programmable Network.* PhD thesis, University of Cambridge Computer Laboratory, 2000.

[MHN01] Jonathan T. Moore, Michael Hicks, and Scott Nettles. Practical programmable packets. In *Proceedings of the Twentieth IEEE Computer and Communication Society INFOCOM Conference*, pages 41–50. IEEE, April 2001.

[MMR98] W. S. Marcus, A. J. McAuley, and T. Raleigh. Protocol Boosters: A Kernel-Level Implementation. In *Proceedings of IEEE Globecom*, November 1998.

[Moo02] Jonathan T. Moore. *Practical Active Packets*. PhD thesis, University of Pennsylvania, September 2002.

[MS98a] S. J. Muir and J. M. Smith. AsyMOS: An Asymmetric Multiprocessor Operating System. In *Proceedings, 1st OpenARCH Conference*, pages 25–34, April 1998.

[MS98b] S. J. Muir and J. M. Smith. Supporting continuous media in the Piglet OS. In *The 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 99–102, July 1998.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. The MIT Press, 1990.

[Mui01] Steve J. Muir. *Piglet: An Operating System for Network Appliances*. PhD thesis, CIS Department, University of Pennsylvania, 2001.

[Nee94] Roger M. Needham. Denial of Service: An Example. *Communications of the ACM*, 37(11):42–46, November 1994.

[NML98] Peter Newman, Greg Minshall, and Thomas Lyon. IP Switching – ATM Under IP. *IEEE/ACM Transactions on Networking*, pages 117–129, April 1998.

[NOG93] Scott M. Nettles, James W. O'Toole, and David Gifford. Concurrent garbage collection of persistent heaps. Technical Report MIT/LCS/TR–569 and CMU–CS–93–137, Computer Science Department, Carnegie-Mellon University, April 1993.

[Par92] Craig Partridge. *Late-Binding RPC: A Paradigm for Distributed Computation in a Gigabit Environment*. PhD thesis, Harvard University, 1992.

[PJ96] Craig Partridge and Alden Jackson. Smart packets. Technical report, BBN, 1996. `http://www.net-tech.bbn.com/smtpkts/smtpkts-index.html`.

[PSS+00] C. Partridge, A. Snoeren, T. Strayer, B. Schwartz, M. Condell, and I. Castineyra. FIRE: Flexible Intra-AS Routing Environment. In *Proceedings, ACM SIGCOMM Conference*, pages 191–203, 2000.

[Rit84] D.M. Ritchie. A stream input-output system. *AT&T Bell Laboratories Technical Journal*, 63(8):1897–1910, October 1984. Part 2.

[SFG+96] J. M. Smith, D. J. Farber, C. A. Gunter, S. M. Nettles, D. C. Feldmeier, and W. D. Sincoskie. Switchware: Accelerating network evolution (white paper). Technical report, University of Pennsylvania, `http//www.cis.upenn.edu/~jms/white-paper.ps`, June 1996.

[SG90a] James W. Stamos and David K. Gifford. Implementing Remote Evaluation. *IEEE Trans. Software Engineering*, 16(7):710–722, July 1990.

[SG90b] James W. Stamos and David K. Gifford. Remote Evaluation. *ACM Trans. Programming Languages and Systems*, 12(4):537–565, October 1990.

[SGQM88] Jonathan M. Smith and Jr. Gerald Q. Maguire. Process Migration: Effects on Scientific Computation. *ACM SIGPLAN Notices*, 23(3):102–106, March 1988.

[SI89] Jonathan M. Smith and John Ioannidis. Implementing remote *fork()* with checkpoint/restart. *IEEE Technical Committee on Operating Systems Newsletter*, pages 12–16, February 1989.

[Sin02] W. David Sincoskie. Broadband Packet Switching: A Personal Perspective. *IEEE Communications Magazine*, 40(7):54–66, July 2002.

[SJS+00] Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, Dennis Rockwell, and Craig Partridge. Smart Packets for Active Networks. *ACM Transactions on Computer Systems*, 18(1):67–88, February 2000.

[Smi88] Jonathan M. Smith. A Survey of Process Migration Mechanisms. *ACM SIGOPS Operating Systems Review*, July 1988.

[SON91] *Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, Report TR-NWT-000253*. Bellcore, December 1991. Issue 2.

[SPS+01] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakuontio, Stephen T. Kent, and W. Timothy Strayer. Hash-based IP Traceback. In *Proceedings, ACM SIGCOMM Conference*, pages 3–14, 2001.

[SSF99] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: A Fast Capability System. In *Proceedings, 17th SOSP*, pages 170–187, Dec 1999.

[Sta86] James W. Stamos. *Remote Evaluation*. PhD thesis, MIT, January 1986.

[Sta90] *Computer* Staff. Gigabit network testbeds. *IEEE Computer*, 23(9):77–80, September 1990.

[SWKA00] Stefan Savage, David Wetherall, Anna R. Karlin, and Tom Anderson. Practical Support for IP Traceback. In *Proceedings, ACM SIGCOMM Conference*, pages 295–306, 2000.

[Tan88] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, second edition, 1988.

[TSS+97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications*, 35(1):80–86, January 1997.

[TW96] David Tennenhouse and David Wetherall. Toward an Active Network Architecture. *Computer Communications Review*, 26(2), 1996.

[vdML98] J. E. van der Merwe and I. M. Leslie. Service-Specific Control Architectures for ATM. *IEEE Journal on Selected Areas in Communication (Special Issue on Protocol Architectures for 21st Century Applications)*, 16(3), April 1998.

[Vol72] Francois-Marie Arouet Voltaire. *Contes*. 1772. An Italian proverb 'La meglio e l'inimico del bene' quoted by Voltaire.

[Wal82] David Wayne Wall. Messages as Active Agents. In *Proceedings, 9th Annual POPL*, pages 34–39, 1982.

[WGT98] David J. Wetherall, John Guttag, and David L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *Proceedings, IEEE OpenArch 98*, pages 117–129, 1998.

[Yd96] Y. Yemini and S. daSilva. Towards Programmable Networks. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1996. `http://www.cs.columbia.edu/~dasilva/netscript.html`.

[ZF80] J. Zander and R. Forchheimer. Preliminary Specification of a Distributed Packet Radio System Using the Amateur Band. Technical Report LiTH-ISY-I-408, University of Linkoping, January 1980.

[ZF83] J. Zander and R. Forchheimer. Softnet - An Approach to Higher Level Packet Radio. In *Proceedings, AMRAD Conference*, 1983.

9 Appendix 1 - The Store, Translate and Forward Model

Store, Translate and Forward (STF) networks

ABSTRACT
Store and Forward networks provide a rich framework within which network technologies, such as packet store and forward networks have evolved. Examples include the Internet and the developing Asynchronous Transfer Mode (ATM) networks.
A major problem with the communications technologies we have developed is their incompatibility at the level of service. This level is the point at which the communications become ‘‘data communications’’ rather than sensory data or facsimiles. This interservice networking promises to become a major research and engineering challenge in the mid- to late 1990s and on into the 21st century.
We propose a model of networking we call STF, for store translate and forward. The network is modeled as a graph. Nodes store and forward messages, and arcs may be associated with a translation. We are hopeful that ideas from computer programming languages can aid with the design of data typing and translation strategies. We give several examples of STF scenarios and possible translation modules. We also demonstrate analogues to other technical problems, such as heterogeneous distributed shared memories.
We believe that STF is a promising approach to interservice networking. It selectively borrows from a wide range of disciplines such as programming languages, computer networking, and operating systems. Most importantly, by explicitly embedding translation into store and forward networking, it recognizes the presence of processing elements in communications systems.

10 Appendix 2 - Applications of Store, Translate and Forward Networks

Some applications of S-T-F Networks:

1. Combining Cellular Telephony/PCX with GPS locators to provide a global mapping capability. GPS sensors pick up GPS data. These are "translated" into a form suitable for transmission to cellular, then across landlines into a computer fabric equipped with DSM. The object location facilities embedded in the DSM are then used to make a map, zoom, combine data with Landsat data to get real-time object movement, etc.

2. Message conversion to meet changing QoS goals across communications boundaries. Consider, for example, a need to carry a message from a communications source, across secure, reliable landlines to a relay point. From the relay point, however, the reliability of any single means of communication (LEOS, point-to-point radio, telephony,....) is low. At the relay point, an STF network could replicate message packets across several alternatives to build a reliable subsystem from unreliable components.

3. Provable location information. Consider a STF network organized as a grid, say a planet-girdling grid of appropriate granularity. Longtitude and Latitude might make an appropriate coordinate system. As packets cross grid boundaries, a token they contain is tranformed using a key- controlled cryptographic system. The token can then be used to prove a particular path was taken, and that in turn, the packet’s source location can be proven.

4. Heterogeneous Distributed Shared Memory. One of the major difficulties with DSM has been the inability to effectively use it in networks of heterogeneous machines. In fact, machines of the same type could be organized into "clusters", which define boundaries over which translation of data would be coupled with forwarding. In this way, the important information (tagged, e.g., as ASN.1) could cross the boundary and be reconstituted in the new cluster environment.