Reflections on Network Architecture: an Active Networking Perspective

Ken Calvert[*]
Laboratory for Advanced Networking
University of Kentucky
calvert@netlab.uky.edu

ABSTRACT

After a long period when networking research seemed to be focused mainly on making the existing Internet work better, interest in "clean slate" approaches to network architecture seems to be growing. Beginning with the DARPA program in the mid-1990's, researchers working on *active networks* explored such an approach, based on the idea of a programming interface as the basic interoperability mechanism of the network. This note draws on the author's experiences in that effort and attempts to extract some observations or "lessons learned" that may be relevant to more general network architecture research.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Packet-switching networks; C.2.2 [**Network Protocols**]: Protocol Architecture

General Terms

Design, Experimentation

1. INTRODUCTION

Networking is a complex, cooperative enterprise; as such it requires division of labor and a clear assignment of functional responsibilities. A well-designed *network architecture* provides an organizational framework that is more or less independent of specific technologies and applications. Although the protocol architecture of the Internet has undoubtedly been a key factor in its success, it is not perfect, and indeed its shortcomings are becoming more evident. During the 1990's networking research seemed to take place mainly (or even exclusively) within the framework of the TCP/IP protocol architecture, with the worthy goal of making the existing Internet work better. Recently, however, as the shortcomings of the Internet architecture become more evident, we have seen renewed interest in network architecture from some quarters, including funding agencies [5, 12].

Beginning with the DARPA program in the mid-1990s, a group of researchers considered a "clean slate" approach to network architecture, based on the idea of providing network service via a *programming* interface, as opposed to a static packet format [17]. Although the term "active network" came to be associated with one particular approach—in which each packet could carry instructions specifying its own handling—various dimensions of the concept related to network architecture were explored over the last 10 years. As such, the work on active networking represents one of the few recent research efforts that considered network architecture as something other than a solved problem.

This note is an attempt to present some observations based on the author's experiences, in the hope of extracting some insight into network architecture. It is neither an attempt to make a case for active or programmable networking, nor to claim that experience with active networks (AN) architecture *necessarily* transfers to other any other kind of network architecture. The AN effort, after all, focused on solving a particular problem—the difficult and lengthy process of standardizing and deploying new protocols and services—using a particular technology: programmable network nodes. Nevertheless, there may be some general lessons here. The reader will judge whether that is the case.

2. ACTIVE NETWORK ARCHITECTURE

The development of the DARPA Architectural Framework for Active Networks [7] was motivated by the need to reach agreement about assumptions and goals for the DARPA active networks research program. Although there had been a fair amount of work on active networking architecture up to that time [1, 18, 19], there was a need for a common vocabulary and frame of reference for researchers in the program to understand, and, especially, leverage each others' work. The DARPA framework was never an attempt to define or prescribe any kind of standard. Rather, its aim was to identify components and interfaces of a generic "network node platform," and suggest a reasonable assignment of functionality so that work on different aspects could proceed independently. A goal explicitly stated in [7] is to provide a model sufficiently general to accomodate all the work going on in the DARPA program, as well as experimentation with other, perhaps even more radical, paradigms. Among other things, this meant that the architecture needed to accomodate both the "capsule" model, in which programs, or references to programs, are carried instead of standardized headers in every packet, and the "programmable router" model, in which the data forwarding path is customizable, but only via an out-of-band interface. Another design goal of the architecture was to minimize the amount of global agreement required among communicating entities. The result was a minimal framework, rather than a complete network architecture. The architectural framework specifies the functional components of a network *node*, rather than any particular end-to-end service. The idea was that once the right set of functions was available in a programmable node platform, a variety of end-to-end services could (and would) be built.

The hierarchy of components at a node is shown in Figure 1. Each node in the active network runs one or more *execution environments* (EEs), each of which defines a "virtual machine" that

[*]The support of DARPA, Intel Corporation, Air Force Research Laboratory, and the National Science Foundation, and the contributions of many collaborators and students are gratefully acknowledged. Opinions expressed here are solely those of the author.

operates on packets. (For example, an EE might be a Java Virtual Machine, suitably extended to parse bytecode programs carried in packets, and send running code as packets.) Users invoke *Active Applications* (AAs), which provide code to program an EE to implement an end-to-end service. Execution environments gain access to node resources (computing and transmission bandwidth, storage) via a *Node Operating System* (NodeOS), which is responsible for managing/sharing those resources among EEs at the level of the node.

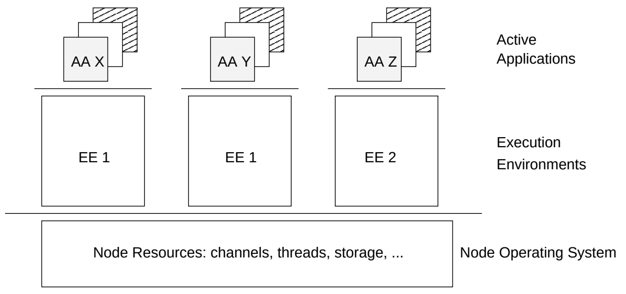

Figure 1: Functional Components of Active Node Architecture

Each EE provides a set of abstractions from which end-to-end services can be composed; each AA realizes such a service using the set of abstractions provided by an EE. Thus, each AA is designed (and constrained) to operate in the context of a particular EE. The EE abstractions, in turn, are built from the basic elements provided by the NodeOS, which hides the particulars of the hardware platform. The NodeOS also provides basic "plumbing" facilities to allow packets to be directed to/from EEs.

An important aspect of traditional network architectures is the identification of the set of protocols that specify the form and meaning of the "bits on the wire". In the active network architecture, the wire formats are determined by the EE, by any underlying "legacy" channel protocols, and by the Active Network Encapsulation Protocol. ANEP [10] is in principle the only aspect of the AN architecture that needs to be standardized. It provides a mechanism, consistent across all nodes, that enables EEs and packets to rendezvous: by placing appropriate values in the ANEP header of their packets, users can arrange for them to be processed by the EE of their choice.

The framework evolved over time. The first version identified only the NodeOS and EEs as components; the notion of an active application came later, as the focus moved from individual node architecture to end-to-end services based on node capabilities. Originally the framework document described the NodeOS programming interface in detail. At some point the specifics of the programming interface were relegated to a separate document so that they could evolve without affecting the overall architecture, while the description of the resource abstractions (thread pools, memory pools, channels, etc.) remained. In the final version, all mention of particular resources managed by the NodeOS was removed, leaving the description completely neutral with respect to the NodeOS programming interface [8, 15]. In a similar way, the security-relevant portions of the architecture, which were originally treated (somewhat sketchily) in the framework document, became a more comprehensive document that stood on its own [9].

3. SOME OBSERVATIONS

Here, in more or less random order, are a handful of observations. Some are technology-specific, others have more to do with the nature of research in network architecture. Each reflects the evolution of some aspect of at least one person's architectural thinking.

3.1 Hardware and Packet-Processing Context

Modern routers typically process packets on *port cards*, which connect to a channel on one side and a high-capacity interconnect on the other (Fig. 2); the interconnect simply transfers packets from one port card to another. This structure, which seems to be inherent in the topology of the problem, has the following consequences. First, the processing on one port does not—indeed, cannot—interact directly with the processing on other ports. The router is basically a parallel multiprocessor. Second, processing that occurs on a port card only needs to keep up with the channel; per-packet processing that occurs in a central location must be able to keep up with the aggregate speed of all channels to avoid overload. Third, packet-processing occurs in two contexts: the input port context and the output port context. Routers may also have a third context: a separate, general-purpose processing environment that takes care of packets that require special handling, such as routing protocol packets or those addressed to the router itself.

The functional node structure depicted in Figure 1, on the other hand, looks more like that of a general-purpose computer than a router. (Jonathan Turner first pointed this out to me.) This is not accidental; as a tool for separating concerns in an endeavor concerned primarily with *programming*, the structure is sensible and useful. Moreover, prototype active nodes—as is typical in many networking research projects—were mostly implemented on general-purpose PC platforms, with off-the-shelf operating systems (linux) providing the resource management and protection functions needed between EEs implemented as user processes. Of course PC-based nodes could never provide high performance (even for conventional networks), but the AN effort was not primarily about performance anyway.

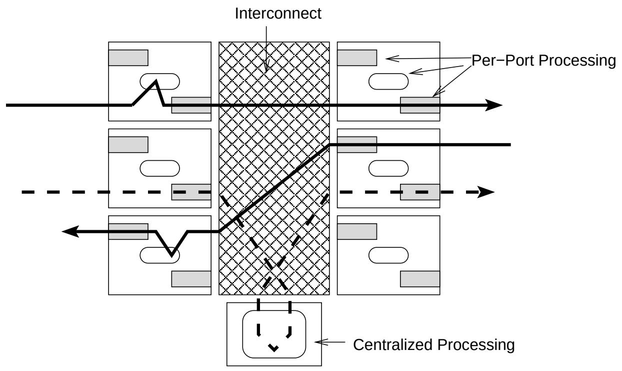

Figure 2: Processing contexts in a router

The challenge is to ensure that Figure 1's view of the system does not obscure the notions of context that will become important when it must be mapped to Figure 2. In particular, in designing an EE (or for that matter any per-hop processing in a packet network), the appropriate context for each function needs to be considered. Whether this is straightforward or not depends very much upon the EE's packet-processing model. If the model specifies a sequence of processing steps applied to each packet, the transfer of the packet from the input port to the output port via the interconnect should be explicitly identified in the sequence, to serve as a demarcation point between the input and output contexts. The CANEs EE, for example, did not identify such a point, although it would not have been difficult to add it [2]. Other, language-based EEs also ignored the distinction [20, 14], perhaps because they never considered that the system would be implemented on a hardware architecture like that of Fig. 2.

EE designers considering such a split would need some mechanism for bundling up the per-packet EE state and transferring it along with the packet from input to output context. Since the underlying hardware architecture would strongly influence such a state-transfer function, the NodeOS should provide it as a service. However, the NodeOS API never included such a function. The moral of the story is that processing *context* should be considered when specifying the general packet-processing model of a forwarding node and the mechanisms to be provided within that model.

3.2 Node vs. End-to-End Focus

One of the selling points of programmable networks is the potential to change the end-to-end network service "on a dime". In order to avoid constraining the set of possible services, the DARPA framework explicitly avoided saying much about the *end-to-end* architecture, except that it was packet-based. Instead, it focused on the basic building-block of the network, the node platform. Empirically it seems that there has been a good deal more research on platform-oriented building blocks (NodeOS and EEs) than on the services that can be built from them. A simple explanation for this is that node-based building blocks are necessary for the end-to-end services, and the whole point of the DARPA program was to produce the *platform*, rather than any set of particular end-to-end services. Yet this proportion seems to have persisted over a long time. As one example, consider that a number of programmable network platforms were defined that offered a wonderful opportunity to investigate end-to-end issues considered important today, such as routing and addressing. Alas, few, if any, published results in those areas use active networks as a vehicle.

The general observation is that it is much easier to get solid results when research focuses on some *local* aspect of a network, as opposed to the total end-to-end system. The difficulty of getting definitive results about global end-to-end approaches is well known: it is hard to provide an experimental infrastructure that is both realistic in scale *and* controllable. Even given such an infrastructure, it is far from clear how to set parameters governing, e.g., the nature and amount of background traffic; the parameter spaces are simply far too large to explore thoroughly. (See Floyd and Paxson [11] for an excellent discussion of these issues with respect to simulation.) In contrast, research on local problems and mechanisms—such as fair scheduling algorithms, efficient switching architectures, packet classification algorithms, and even TCP congestion control mechanisms—generally requires modest infrastructure, and can be conducted in a controlled environment or analyzed with (relatively) simple models. This is not to say that such research is in any way less valuable, but rather that it can often provide more "bang for the buck".

3.3 Late Binding: Boon or Bane?

The DARPA framework attempted to keep as many options open as possible by not specifying any particular programming model, instead allowing different models (in the form of different EEs) to "compete". The architecture specified a set of basic functions, but left to the community the specification of a network API that would provide access to those services. Several such APIs (EEs) were defined [20, 14, 2], but no single one ever really emerged triumphant. One can imagine several explanations for this. One possibility is that all of the defined APIs were deficient in some way. Another is that the research program ended before any approach could gain sufficient momentum. A third is that there was just not enough demand for network programmability to allow differentiation among approaches. However, I believe there is a principle involved that applies to network architecture more generally.

The ability to support a variety of *policies* is clearly a desirable goal for any network architecture. Certainly many of the recognized shortcomings of today's Internet relate to the desire of various stakeholders to enforce policies, where there is inadequate mechanism to do so. For example, an architecture designed for late binding of, say, policies regarding which packets will be allowed to transit through a domain (as in [16]) would eliminate problems arising from conflicting provider policies in BGP [13]. On the other hand, allowing late binding of *mechanism* (i.e. providing multiple ways to achieve the same thing) can actually *hinder* interoperability. In active networks the question inevitably arose whether two hosts that supported disjoint sets of EEs would be able to communicate; shortly after that came the notion of processing a packet through multiple EEs on the same node.

There are other, well-known historical illustrations of this phenomenon. The OSI network layer featured both a connectionless and a connection-oriented service; they did not interoperate. In contrast, though flexibility of service was one of the primary goals of the Internet architecture [6], it was designed to achieve that goal through a single mechanism: best-effort datagram forwarding, the "waist of the hourglass".

It is true that some situations *require* flexibility of mechanism. Cryptographic algorithms in protocols are the canonical example: late binding is necessary because of the potential for any given algorithm to be "broken" at any time. In other situations (routing algorithms), there is advantage in using different mechanisms in different scopes. But in many cases, late binding of mechanisms is simply an admission by the architects that they could not pick a "winner" among the possible mechanisms. So they call it a draw, perhaps declaring that things will be sorted out in the future by "competition." As observed above, the situation in networking is different from that in graphics or CPU architecture. Competition among network mechanisms can *raise* the total cost to the user, because multiple mechanisms must be supported to achieve universal connectivity.

A related point is what might be called "the tyranny of the clean slate". It would seem that a fresh start like that provided by a programmable infrastructure would make re-inventing the network layer easier, by allowing new assumptions and exploration of new tradeoffs. The unfortunate reality is that starting with a clean slate is *harder* than an incremental approach, because so many things have to be considered at once. The space of possible solutions is so vast that one cannot always foresee the results of tradeoffs without significant experimentation. Research projects with limited resources must focus on a few specific aspects of the architecture; for other aspects they end up adopting existing, known-to-work solutions in order to avoid re-inventing (or just re-implementing) too many wheels. The result is often something that is mostly incremental, even if the slate was clean initially.

The moral of the story is that constraints on the solution space are a good thing, because they provide focus [3, 4].

3.4 Internal Interfaces: Specified vs. Implicit

The NodeOS Interface Specification [8, 15] ended up being based on POSIX-like capabilities. The original motivation for specifying this interface within the group was so that EE developers could count on a basic level of functionality to be available at every active node. Given such a consistent interface, porting EEs between different active nodes would be a simple matter.

This approach had some interesting consequences. First, it implied that the primary burden of EE (and thus, in fact, active network) deployability fell on the NodeOS implementors: an EE developed to the NodeOS interface specification could not be deployed on a platform until that platform supported the interface. Second, in order to quickly reach agreement on a basic specification, radical, controversial, or platform-specific functions were simply left out of the drafts. Indeed, the NodeOS specification says that where non-AN-specific functionality is needed, POSIX functions should be used if possible. In other words, the NodeOS Interface Specification was rather biased toward the functionality provided by a general-purpose computing environment.

An alternative approach would have been to agree on the *existence* of the interface, and the separation of concerns it implies, while leaving the details of the interface to the node "vendor." This puts the burden of adapting the EE's model to each vendor's programming interface entirely on the EE developer. In retrospect, it seems likely that node developers would provide a sufficiently large set of common functions to enable an EE to be ported to all platforms.

The lesson is that specification of internal interfaces should come last, rather than first, and internal interfaces should be specified only when there is a compelling reason to do so. The IETF has a long-standing tradition of specifying only the "bits on the wire", and letting internal interfaces take care of themselves. This provides maximum freedom to the implementor, and allows a more direct "competition" of functionalities vying for inclusion.

3.5 Proving "Lifecycle" Benefits

The ability to more easily and quickly upgrade and deploy services has often been cited as motivation for active/programmable networks. Moreover, improved managability and control is likely to be an important goal for the next generation Internet architecture. Unfortunately, to demonstrate or rigorously quantify such benefits, a very long baseline is required. Since nobody really knows the half-life of a protocol architecture deployed on millions of nodes, it is hard to even estimate the time required. How large of a network is required before management costs of a proposed architecture could be fairly compared with costs in the existing Internet? It seems that the only way to for a new architecture to achieve acceptance is for it to offer obvious advantages from the outset—for example, freedom from denial-of-service attacks.

4. CONCLUSIONS

Much of the foregoing now seems "obvious in hindsight", but it must be noted that the author's views have changed on every single one of the above issues, in some cases rather dramatically. That may simply reflect the meager state of my understanding at the beginning. Perhaps a more important conclusion is that research in network architecture is just hard.

5. REFERENCES

[1] D. Alexander, W. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, J. Moore, C. Gunter, S. Nettles, and J. Smith. The Switchware Active Network Architecture. *IEEE Network Special Issue on Active and Controllable Networks*, 12(3):29–36, 1998.

[2] S. Bhattacharjee, K. Calvert, Y. Chae, S. Merugu, M. Sanders, and E. Zegura. Canes: An execution environment for composable services. In *DARPA Active Networks Conference and Exposition*, pages 255–272, May 2002.

[3] K. Calvert, J. Griffioen, B. Mullins, A. Sehgal, and S. Wen. Concast: Design and implementation of an active network service. *IEEE Journal on Selected Areas of Communications*, 19(3), March 2001.

[4] K. Calvert, J. Griffioen, and S. Wen. Lightweight network support for scalable end-to-end services. In *ACM SIGCOMM 2002*, pages 265–278, August 2002.

[5] D. Clark et al. New arch: Future generation network architecture (final technical report), 2004. http://www.isi.edu/newarch/iDOCS/final.finalreport.pdf.

[6] David D. Clark. The design philosophy of the DARPA internet protocols. In *ACM SIGCOMM '88*, pages 106–114, August 1988.

[7] K. L. Calvert (editor). Architectural Framework for Active Networks. DARPA Active Networks Working Group Draft, December 2001.

[8] L. L. Peterson (editor). NodeOS Interface specification. DARPA Active Networks Working Group Draft, January 2001.

[9] S. Murphy (editor). Security Architecture for Active Nets. DARPA Active Networks Working Group Draft, November 2001.

[10] D. Scott Alexander et. al. Active Network Encapsulation Protocol (ANEP), 1997. http://www.cis.upenn.edu/ dsl/switchware/ANEP/.

[11] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001.

[12] National Science Foundation. Future internet design program. http://find.isi.edu.

[13] T. Griffin and G. Wilfong. An analysis of BGP convergence properties. In *ACM SIGCOMM '99*, pages 277–288, September 1999.

[14] Michael W. Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A packet language for active networks. In *International Conference on Functional Programming*, pages 86–93, 1998.

[15] L. Peterson, Y. Gottlieb, M. Hibler, P. Tullmann, J. Lepreau, S. Schwab, H. Dandekar, A. Purtell, and J. Hartman. An OS Interface for Active Routers. *IEEE Journal on Selected areas of Communicasitons*, 19(3), March 2001.

[16] B. Raghavan and A. Snoeren. A system for authenticated policy-compliant routing. In *ACM SIGCOMM 2004*, pages 167–178, August 2004.

[17] J. M. Smith, K. L. Calvert, S. L. Murphy, H. K. Orman, and L. L. Peterson. Activating networks: A progress report. *IEEE Computer*, 32(4), April 1999.

[18] David L. Tennenhouse and David J Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2), April 1996.

[19] Christian F. Tschudin. The Messenger Environment M0 - a Condensed Description. http://cui.unige.ch/tios/msgr/m0/doc/overview.html, May 1997.

[20] David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH '98*, April 1998.