

Towards Application-Private Networks (APNets)

Dekai Li and Jonathan M. Smith
CIS Department, University of Pennsylvania

February 5th, 2003

Abstract

Application-Private Networking is a new approach to network architecture which chooses protocol elements, including multiplexing strategies, based on application requirements. The approach fills the gap between programmable networking approaches such as active networking and the requirements of networked applications. The application's requirements are declarative, and to the degree possible, application-centric choices of protocol elements are made. As both network conditions and application requirements vary, the protocol architecture itself is dynamic.

Completely avoiding scaling issues or boundary concerns, we have implemented a proof-of-concept software architecture useful in both hosts and routers called "Multiple restricted Virtual Machine" or MrVM. We demonstrate two applications, a LAN-optimized NFS and an application-directed multicast, that configure a networking system on the basis of specified application requirements.

We close with some speculation on how APNets, combined with improved data on network conditions (a "Knowledge Plane"), could transform the future Internet to be *both* more usable and more secure.

1 Introduction

The design space for network architectures can be conveniently described as a 3-tuple of $\langle \textit{Application Requirements}, \textit{Protocol Elements}, \textit{Network Conditions} \rangle$. *Application requirements* can range from reliability and small message interarrival delay to communications secrecy. *Protocol elements* include multiplexing schemes such as circuit-switching and packet-switching, acknowledgement schemes and error-correcting codes, timers and a plethora of cryptographic transformations. *Network conditions* include delay, delay variance, loss rates, bit error rates (BERs), topology, and available bandwidths. For any given triple, and in particular for a choice of application and requirements, there are assumptions about operating conditions made (although these assumptions may be obscured by layering). Protocol elements are selected to meet the application requirements under these operating conditions.

Two examples, the telephone network and the Internet, are

useful in understanding this architectural framework.

The telephone system's network architecture [1] is engineered to deliver a band-limited audio channel appropriate for interactive voice telecommunications. The application requirements then, include, the ability to deliver about 3000Hz of audio, with some limits on delay and audible impairments; these requirements have been met in the telephony architecture by using a call set-up protocol (of considerable complexity) to set up a point-to-point channel used to carry a voice stream. Link, multiplexor, switch and capacity engineering are voice-centric.

The Internet design [7, 10], requiring interoperation across a variety of networks and operating conditions, and intended to service many applications [9], must choose protocols that can tolerate an extremely wide variety of network conditions. Thus, the basic IP transport service is a minimal datagram service, response to network dynamics such as topology changes is provided by dynamic routing. Other, more application-specific requirements such as ordering, reliability, *etc.* are provided by end-to-end overlay protocols such as the Transmission Control Protocol, TCP.

If we contrast the Internet architecture with the telephony network architecture, TCP/IP is intended to be agnostic with respect to applications, and adapts to a large (but not all-encompassing) range of network conditions with its choice of protocol elements. To optimize the placement of protocol functions in the architecture (as opposed to for a specific requirement) the "end-to-end" [26] design notion pushes functions to the end-points, eliminating redundant implementation and giving application designers the widest range of options for use of the basic network service.

These two examples illustrate the design space and tradeoffs made amongst its "dimensions". Neither architecture is ideal - for example the attempt to remove many dynamics in network conditions within the call makes the telephony architecture limited in its ability to efficiently handle applications with dynamics very different than that of voice. Likewise, the IP architecture's engineering approach to dealing with many applications and network conditions has forced engineering tradeoffs, such as substantial overprovisioning [20] (to control delay jitter) to support applications such as voice and video.

The next section, Section 2 presents an abstract design target for an ambitious network research agenda, the *Application-*

2 Automated Optimal Network Engineering

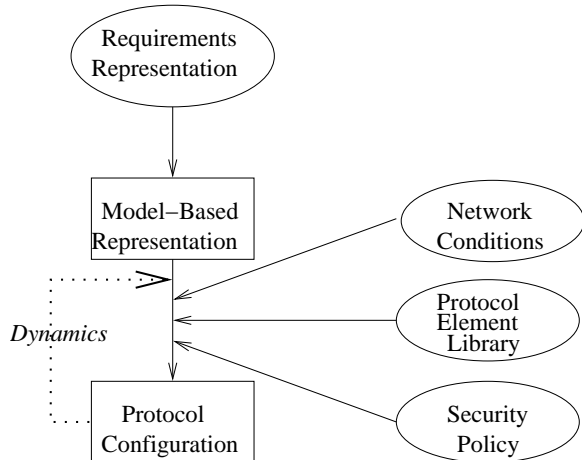


Figure 1: Dynamic Protocol Refinement

An ideal network architecture within the constraints of our design space would have the property that at any given time, the application requirements and network conditions would result in the best known selection and placement of protocol elements. For example, if network condition dynamics result in a variable BER, as in a mobile wireless context, the protocol architecture might be adjusted to inject forward error correction (FEC) to move TCP/IP into an operating regime where its protocol element selections result in meeting application requirements. 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.

While limited instances of such techniques have been demonstrated experimentally [14], the ideal system would automate [21] such responses, under control of high-level models of application requirements. A great deal of detail is masked by the design space abstraction of Section 1 but the basic point is not to be lost: for any specified application requirements (including preferences, weights, etc.) and network conditions (we will discuss how information about such network conditions might be made available using the “Knowledge Plane” [8] proposed by David Clark of MIT, in Section 3), one or more equivalent selections of protocol elements can be made which closely meet the application requirements.

As this process is fundamentally driven by application requirements, we call such networks *Application-Private Networks*, or *APNets*. The basic design process for an APNet for

a particular application would result in a protocol architecture optimized for that application’s performance, with protocol elements selected in concert with any techniques, such as time-division multiplexing, needed to limit the range of network conditions for these selections. The resulting network architecture is colloquially a “stovepipe”.

An excellent example design from the space systems domain is the “Remote Agent” [21] architecture used in NASA’s Deep Space One (DS1) mission, where many of the challenges are similar to those of network engineering, such as multiple timescales, unplanned events, and overall “mission goals”. In the NASA system, very high-level models are used to drive a planning system; current conditions are fed into a system with a limited time horizon to drive specific actions such as recovery, reconfiguration and reprogramming in the face of system conditions such as failed sensors and actuators.

The challenge in the more general case is large-scale sharing. That is, “stovepipe” design is economically inefficient, inhibits adaptation and reuse, and makes interoperability with other applications, as well as sharing of facilities, difficult. Further, it makes unfounded assumptions for the general case, where conflicting goals among users are common. The advent of programmability in many network components, such as network processors, software radios and extensible routers, permits the configuration of such components to be virtualized. That is, the component behavior can support multiple application-driven specializations. The problem is not easy, but is conceptually within reach [21], as demonstrated by the DS1 experiments we have discussed. An abstraction is given in Figure 2, where application requirements (specified, perhaps as in Section 3) induce behaviors at various logical levels in a network, from host to link.

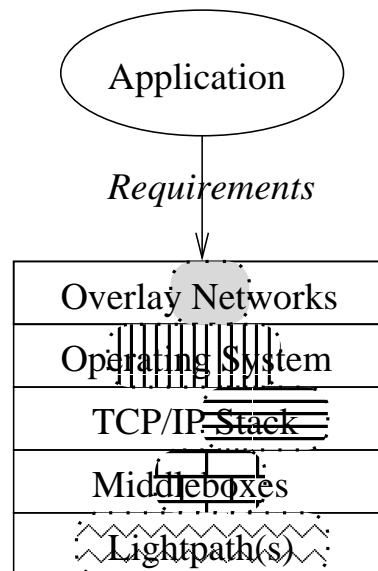


Figure 2: APNet Construction

This process will take place repeatedly based on changes in network conditions. The reconfiguration process must be safe, network knowledge must be available to both the protocol element selection and programmable component configuration processes, and the network knowledge must be trusted, to deal with accidental and malicious failures.

Among the interesting technical questions to be resolved are issues of security, stability and degree of extensibility for the architecture as a whole. To touch just briefly on these issues, the degrees of extensibility might include those possible from a machine learning algorithm in optimization of protocol selections, they might include addition of new protocol elements as they are discovered, or they might include wholesale changes of the control architecture itself. Stability issues include overreactions, damping and convergence of distributed control schemes. Prototyping and experiments can identify the appropriate adaptation rates for various timescales, ranging from the immediate to relatively long-term, which some researchers have categorized as *reactive*, *deliberative* and *reflective* - these adaptation timescales will affect the dynamics of APNet instances. Security concerns, in addition to the trust of network condition data, include the risk of subtle Denial of Service attacks on a complex infrastructure, data privacy, authorization for code loading, provenance of aggregated data, and finally, the technically difficult issue of what the telephony industry refers to as "feature interaction".

3 Network Knowledge Representation

Clark's "Knowledge Plane" [8](KP), while not yet completely defined, is conceptually a repository for knowledge about the network, in the same sense that a control plane is used to signal and control data flow in the network. Example roles for the knowledge plane would be to support *diagnosis* (for example, of where a problem is located in the network, without inferring it from the edge in the style of `traceroute`, `ping` or the new "packet-marking" [27, 28] schemes) or *configuration* (for example, of a set of routers airdropped to a remote location). The knowledge contained in the KP, then, might range from CRC error counts on link layer frames, to address and topology information. It crosses essentially all layer boundaries. In addition, the level of trust of knowledge is important.

The interaction between the "Knowledge Plane" and AP-Nets is important, as APNet dynamics will depend on obtaining information from the knowledge plane. If network knowledge is to be widely used it will be named. Much knowledge will be represented syntactically as strings of the form `<name>=<value>`, e.g., "bandwidth=64K". This knowledge representation has been widely adopted for use by applications programmers, in contexts from scripting languages to WWW "cookies", and is readily translated to locally convenient representations. An example use of such a variable is the `TERM` variable maintained by UNIX shells and used to configure ter-

terminal handling in concert with a database of information about terminal capabilities. In an APNet, the host operating system might, using the variables specified by the application, configure schedulers, networking stacks, and choose network adapters.

The string representation enables use of Trust Management [4] technology such as the KeyNote [3] system, which represents assertions as credentials with authorizers, licensees and conditions. Public-key technologies are used to build the web of trust, and a compliance checking process is used to test requested actions against the credentials. Consider public keys for users Alice and Bob, where Alice's key is the licensee, Alice's key is the authorizer, conditions are

```
$file_owner="Alice" && $filename=
"/home/Alice/[^/]*"
&& $hostname="myhost.domain" -> "true";
```

and the signature is with Alice's key. Then Bob is authorized by Alice to access files in Alice's home directory on a particular host at the "domain" domain.

This architecture provides capability-like [23] control of resources and robust delegation of authority in spite of distributed control through its use of cryptography to authenticate and authorize remote operations [22], and has many other desirable features. Complete explication would require more space, but among the desirable properties of credentials and a trusted knowledge plane for advanced applications are: data provenance, support for micro-payment systems of various flavors, authorization for network control, code-loading, resource allocation and digital-rights management. An example use in network architecture might be as authorization to set up a QoS-routed path through the network, with the credential's signature chain leading back to a financial services organization vouching for payment for a premium service.

We have begun exploring the APNet design space by building a software environment with Multiple restricted Virtual Machines (*MrVM*), which allows us to share a host or router among applications with APNet-driven protocol stacks.

4 Exploring the APNet design space with MrVM

In the absence of an operational "Knowledge Plane", an experiment designed to explore the APNets vision must gather information presuming that such a system exists. In practice, this requires both a proposal for knowledge representation, as above, and a multiplicity of sensor/actuator pairs, located in the networking system wherever specialized information about network conditions was available, and adaptation and reactions to this knowledge are required. We sought an experimentally sound design which was realizable without extensive changes in infrastructure, but nonetheless confirmed that the APNets vision itself had merit.

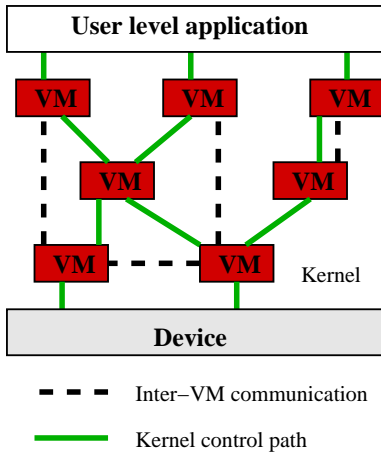


Figure 3: MrVM architecture

Modern applications use a multiplicity of programming environments, with virtual machines (VMs) provided by an operating system as the means of multiplexing the processor(s). Extra-VM system resources are accessed through system call interfaces. We sought a way in which multiple sets of application requirements could be reflected in individualized protocol stacks - and network paths - while successfully interoperating.

Virtual Machines are located in selected control paths in the operating system. These control paths correspond to one set of possible “think points”[8] in a “Knowledge Plane”. We have designed a framework for applications to control a restricted set of activities within the kernel. Application data flows can be processed by a particular VM if code is inserted on behalf of the application. Figure 3 illustrates the architecture.

Our approach to languages and protection is somewhat unorthodox approach, in contrast with the high-level language approaches of, *e.g.*, the SPIN [2] project. Our VM uses a modified DLX instruction set which excludes floating point operations and delayed branches. As a RISC instruction set, it is intended to be efficiently mapped on to real machines (such as network processors, which are similarly register-based), and representative of many embedded processor architectures. One might use a JIT compiler to increase overlapped operation in a system.

Application provided code has its own address space - it is essentially “sandboxed” into a limited portion of the kernel address space. It is thus far more lightweight than a system such as Denali which can sandbox an entire guest operating system. System resources are accessed through MrVM system calls. Applications can put the same code into multiple VMs and use shared memory to implement inter-VM communication.

Since protection is provided by the VM, applications can write code in any programming language and compile it into VM code. In our implementation, we used the LCC compiler from Princeton as the front end, and wrote a back end for the VM architecture. MrVM has been implemented using

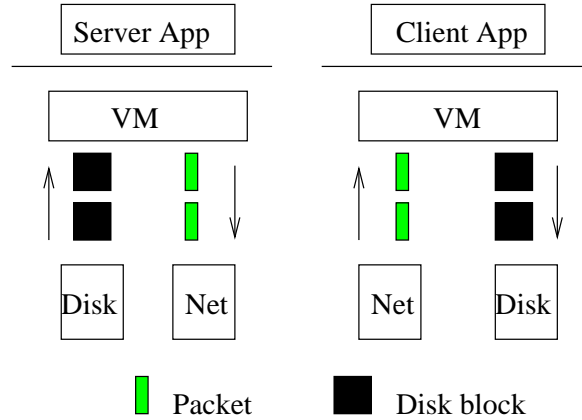


Figure 4: Server and client architecture

the Linux 2.4.20 kernel, with modified Ethernet card drivers.

Strings in the form described in Section 3 are put into the UNIX shell [6] *environment* with `putenv()`, and these values are either used to control the compilation system in producing MrVM code, or are passed along to the VMs as parameters when they are initialized. For example, if a multicast was to be constructed, variables such as “multicast=YES” and “multicast_destinations=10.0.0.23, 10.0.1.37, 10.0.0.145” might be set, and cause the generation of code supporting multicast. The destinations might be dynamically changed in a publish/subscribe manner by the sending application.

4.1 Applications Examples

We have designed two applications of MrVM to demonstrate how this architecture can be used to adapt protocol elements to application requirements in different network environments.

Local File Transfer

For transferring files between hosts local to a LAN, we can avoid substantial work by skipping the IP encapsulation and decapsulation for the data. Rather, MAC addresses are sufficient for file transfer, and we can encapsulate UDP packets directly in Ethernet frames.

We used two VMs to implement the local file transfer application. On the server, the VM sits above the disk I/O system. Upon receiving instructions from the application, it will intercept all disk blocks which would otherwise be copied to the user level space, package these blocks into packets and send them onto the network. The other VM is above the link layer on the client side. It will gather all the packets from the server and write them to the file the client application specifies. This is shown in Figure 4.

To dispatch packets to the correct application, we use the protocol field of the Ethernet header to identify corresponding applications. Thus, this application has some of the

Dst MAC	Src MAC	App Label	App defined header	DATA
---------	---------	-----------	--------------------	------

Figure 5: Packet format

flavor of the “Active Messaging” [30] approach of von Eicken, *et al.* The packet format is shown in Figure 5. For each VM, we partition resources to prevent overuse of shared system resources. Table 1 and 2 show the time to transfer a 25.8MB file and a 113MB file, respectively, between two locally connected hosts on a 100Mbps switched Ethernet. The client machine is an 800Mhz PIII with 384M of RAM and the server machine is a 700Mhz PIII with 256M of RAM. The Ethernet cards were Intel 82559 based 100 Mbps cards.

The comparison has NFS in a cold-cache state. Each VM has 80 network buffers for holding the packets. More CPU time is spent in the kernel for the MrVM-based system because the blocks are manipulated inside the kernel, while NFS will block waiting for packets to come. MrVM performs extra computational work to control the VMs in the kernel to address retransmission and acknowledgement, but performs slightly better than NFS overall due to the “shortcut” taken by packets. The extra CPU cost is a function of our experimental infrastructure, not of the APNets concept.

	Real(s)	User(s)	Sys(s)
MrVM	2.159	0.640	1.500
NFS	2.292	0.0	0.430

Table 1: Transfer time for a 25.7MB file

	Real(s)	User(s)	Sys(s)
MrVM	9.375	2.070	7.250
NFS	9.871	0.010	1.970

Table 2: Transfer time for a 113MB file

Application Controlled Multicast

Application requirements for multipoint communications have stimulated the design of multicast architectures ranging from network layer IP multicast to application level multicast, supported either by the network or directed by the end system. Application controlled multicast to demonstrate how APNets can support certain multipoint communications based on the network conditions within the unicast network. It is a n end system based multicast framework and differs from that of Zhang, *et al.* [15] in that the group members are managed by the sender with the support from MrVM-enabled routers. The receivers are oblivious to details of the multicast.

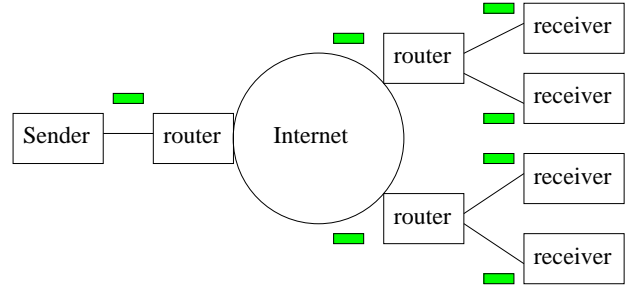


Figure 6: Application controlled multicast

Consider multicast applications such as multi-party conferencing, where packet loss is tolerable, but other requirements are strong, such as bandwidth usage and bounded latency. For example, for typical Cable Modem or ADSL users, the uplink bandwidth may be insufficient to support several flows. In this case (something the “Knowledge Plane” would give us awareness of, *e.g.*, with values such as “Asymmetric=TRUE, UplinkRate=256K, . . .”) we prefer to send a single copy of the packet upstream and let the routers along the path expand the set of recipients. This is shown in Figure 6. The sender sends a copy of the packet to each participating neighbor router, which will forward the packet to downstream participating routers and to the receivers.

When a conferencing session starts, the sender will first send its controlling code along the path to its receiver(s). The sender/upstream router’s IP address is also included in these packets so downstream routers can notify upstream routers when needed. Each MrVM-enabled router will allocate a VM for this session and save a copy of the code. What the code does is identify this session and keep a list of downstream routers/receivers. When packets from the sender come in it will send a copy to each router/receiver in its list.

When a new receiver joins in, the sender will temporarily add its IP address to its list of communicating peers. If the packets along the path are identified by a router, that router will add this address to its list and notify the upstream router/sender to delete that address because that router will copy the packet. This process is repeated until only the router closest to the new receiver has its IP address. All these procedures are performed by the code provided by the sender.

There are two types of control packets. One type informs the upstream router/sender to delete an IP address already in the downstream router’s list. The other type are used by the sender to delete an entry in downstream routers when a receiver quits the session.

The forwarding code is simple and shown below.

```
int n_receivers = 3;
int this_session = 3232;
ip_t receivers[MAX_RCV] = {
    receiver1,
```

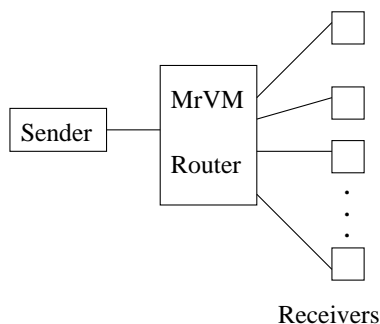


Figure 7: Simple MrVM router

```

receiver2,
receiver3,
}

void MrVM_fw(struct net_buffer *nb)
{
    struct ipheader *ip = nb->ip_h;
    struct session_header *sh = nb->sh;

    if (ip->saddr==SENDER
        &&sh->id==this_session)
        for(i=0; i<n_receivers; i++)
            sendpacket(nb, receivers[i]);
}

```

We used a simple model to test the performance of a MrVM router as shown in Figure 7. The sender injected the code shown into the router. We then varied parameters such as the size of the packets and the number of receivers, and measured the time from the source sending the packet to the time all destinations received it. We used an 800 Mhz PIII with 384M of RAM as the router, and 700 Mhz PIIIs with 256M of RAM for the sender and the receivers. The LAN infrastructure and adapter cards were the same as in the NFS application.

Figure 8 shows the results. Using 512-byte packets as an example, it takes 186 μs for the first packet, and 506 μs for the eighth packet. The average processing time of a packet by the MrVM router is thus $(506 - 186) / 7 = 46\mu s$. If the available bandwidth between the sender and the MrVM router is less than $512 \times 8 \times 10^6 \div 46 = 89 \text{ M bit/s}$, the transmission delay between the sender and the MrVM router will be greater than 46 μs . Thus we can achieve reduced latency for multicast applications using the MrVM router. This will be true for most non-local networks.

4.2 Network Security Considerations

For the local file transfer application, two hosts are presumed connected to the same (bridged) LAN, and the LAN is presumed (whether correctly or not) to be secured. The APNet system can pass MrVM an indication that the local *vs.* non-local distinction might require additional processing, such as

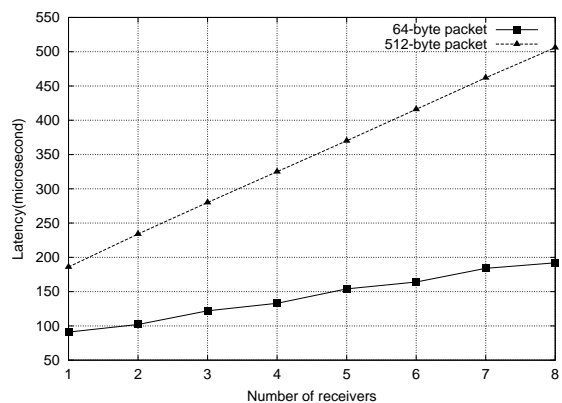


Figure 8: Performance of MrVM router

a pass through IPsec encapsulation. If one did not wish to rely on the LAN presumption, the “Knowledge Plane” might be used to provide a list of trusted *vs.* untrusted hosts which could be used to select whether a privacy transform is required or not.

For the multicast application described previously, the users must be authorized to have access to router VMs, as among other things the number of CPU cycles and network bandwidth they can use are tightly controlled. Authorization is achieved through signed credentials, which are made available through the strings used for APNet knowledge representation. Each VM only has limited access to the physical resources in the router. Malicious users, presumed to be unauthorized, can not use this mechanism to subvert the pool of MrVM VMs, *e.g.*, for use in a network-based DDoS attack.

4.3 Discussion

In the file transfer application, we showed how applications exploit the network conditions and dynamically re-organize the protocol elements. We then extended the use of the MrVM systems from the host-based, local network-based environment to the Internet level to demonstrate that applications can add private support into the existing network, gathering relevant network knowledge at their locations and exploiting it in the service of their applications. The APNet architecture uses available MrVM resources under control of trust managed credentials to provide secure enhanced networking services to applications based on their requirements.

5 Prior and Related Work

Application-centric network concepts were addressed in Clark and Tennenhouse’s architecture paper [11] which introduced the concepts of Application-Level Framing (ALF) and Integrated Layer Processing (ILP). While this paper stimulated considerable work, it was centered on the performance opti-

mizations accruing from reduced memory copying, rather than the implications of an application-driven network architecture.

Network architectures addressing the dynamics of networks discussed in Section 1 have also been explored.

Ritchie's STREAMS [25] architecture provides an elegant design 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-processing modules for streams of data. These modules could be dynamically inserted and removed while the protocol was in operation.

Hutchinson and Peterson's [16] *x*-Kernel 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 would support APNets, but recompilation of an entire protocol architecture would be required to dynamic re-configuration of the protocol graph.

Feldmeier, *et al.* *Protocol Boosters* [12] architecture combines the dynamics of STREAMS with the generalized structure of the *x*-Kernel. Protocol boosters did not address the issue of managing boosted links globally, and did not address the issue of programming environment well. Protocol boosters could be used to support a limited version of APNets as part of an application-driven refinement process as shown in Figure 2.

Active networks [29] have been stigmatized as an architecture in search of an application, but Active Networks are extremely well-suited to the realization of the dynamic protocol architectures of APNets. The combination of the "Knowledge Plane" and the requirements-driven APNets architecture provide an environment in which Active Networks can be exploited.

While not work in network architecture, the "Remote Agent" architecture of NASA's unmanned "Deep Space One" spacecraft has many architectural lessons for APNets. Of particular interest is the way in which a model-based approach is used to deal with mission goals, dynamic reconfiguration is used to deal with exceptions under control of the high-level model, and a real-time executive is used to deal with fine-grained timescales such as those used to manage fuel valves, thrusters and other actuators.

6 Conclusion

Application-Private Networks extend the range of dynamics for protocol architectures, by dynamically selecting protocol elements to meet application requirements in the face of dynamic conditions. Such a network architecture is not only desirable, it is technically achievable within the next decade.

A considerable portion of the architectures (at least the con-

trol plane) of most routers is programmable, leading to a location at which some adaptations for meeting application requirements can occur. For example, the IETF's Forwarding and Control Element Separation (ForCES) working group [13] 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 [24]. New *network processors* [19, 17] are offering a path to wire-speed performance in the forwarding path, 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, and many opportunities for specialized protocol architectures. Finally, wireless systems are introducing technologies amenable to APnet approaches, ranging from mobile telephony industry's use of mobile code to the software definition of radio [18, 5].

A broad range of new network uses are enabled by the dynamic operations of an APNet. We have only just begun exploring the space with the MrVM system, which provides many avenues with which applications can interface with a network element. As various "Knowledge Plane" facilities become available, the role of APNets can increase, as they have the potential to make networks easier to use, network applications easier to write, and to build more trustworthy network infrastructures via credentials.

References

- [1] *Engineering and Operations in the Bell System*. AT&T Bell Laboratories, 2nd edition, 1983.
- [2] B. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Ficzynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety and performance in the spin operating system. In *Proc. 15th SOSP*, pages 267–284, December 1995.
- [3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.
- [4] 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.
- [5] Vanu Bose. *Virtual Radios*. PhD thesis, MIT, 1999.
- [6] S. R. Bourne. The UNIX Shell. *The Bell System Technical Journal*, 57(6):1971–1990, July-August 1978. Part 2.
- [7] V. G. Cerf and R. E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, COM-22, May 1974.

- [8] D. Clark. A New Vision for Network Architecture, September 2002. private communication.
- [9] D. Clark, Scott Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Service Packet Network: Architecture and Mechanism. In *Proceedings, 1992 SIGCOMM Conference*, pages 14–26, August 1992.
- [10] D. D. Clark. The Design Philosophy of the DARPA Internet protocols. *ACM Computer Communications Review*, 18(4):106–114, August 1988.
- [11] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM '90*, pages 200–208, September 1990.
- [12] D. C. Feldmeier, A. McAuley, and J. M. Smith. Protocol Boosters. *IEEE JSAC Special Issue on Protocol Architectures for the 21st Century*, 1998.
- [13] IETF Forwarding Control Element Separation Working Group Home Page. <http://www.ietf.org/html.charters/forces-charter.html>.
- [14] I. Hadžić. *Applying Reconfigurable Computing to Reconfigurable Networks*. PhD thesis, University of Pennsylvania, September 1999.
- [15] Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A Case For End System Multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.
- [16] 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.
- [17] IBM PowerNP Network Processors. http://www-3.ibm.com/chips/products/wired/products/network_processors.html.
- [18] Joseph Mitola III. Software Radios. In *Proceedings, IEEE National Telesystems Conference*. IEEE, May 1992.
- [19] Intel IXP Architecture Network Processors. <http://www.intel.com/design/network/products/npfamily/>.
- [20] Pablo Molinero-Fernandez, Nick McKeown, and Hui Zhang. Is IP Going to Take Over the World (of communications)? In *Proc. Hot Nets Workshop*, 2002.
- [21] N. Muscettola, P. Nayak, B. Pell, and B. C. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1-2):5–48, 1998.
- [22] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks. *Communications of the ACM*, 21(12):993–999, 1978.
- [23] R. Needham and R. D. H. Walker. The Cambridge CAP Computer and its Protection System. In *Proc. 6th Symposium on Operating Systems Principles*, pages 1–10, Nov. 1977.
- [24] 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.
- [25] D.M. Ritchie. A stream input-output system. *AT&T Bell Laboratories Technical Journal*, 63(8):1897–1910, October 1984. Part 2.
- [26] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end Arguments in System Design. In *Proceedings of the 2nd IEEE International Conference on Distributed Computing Systems*, pages 509–512, April 1981.
- [27] Stefan Savage, David Wetherall, Anna R. Karlin, and Tom Anderson. Practical Support for IP Traceback. In *Proceedings, ACM SIGCOMM Conference*, pages 295–306, 2000.
- [28] 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.
- [29] 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.
- [30] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer. Active messages: a mechanism for integrated communication and computation. In *Proc. 19-th International Symposium on Comp. Arch.*, pages 256–266, Gold Coast, Australia, May 1992.