

The Role of Network Processors in Active Networks^{*}

Andreas Kind, Roman Pletka, and Marcel Waldvogel

IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{ank,rap,mwl}@zurich.ibm.com

Abstract. Network processors (NPs) implement a balance between hardware and software that addresses the demand of performance and programmability in active networks (AN). We argue that this makes them an important player in the implementation and deployment of ANs. Besides a general introduction into the relationship of NPs and ANs, we describe the power of this combination in a framework for secure and safe capsule-based active code. We also describe the advantages of off-loading AN control point functionality into the NP and how to execute active code in the data path efficiently. Furthermore, the paper reports on experiences about implementing active networking concepts on the IBM PowerNP network processor.

1 Introduction

The ongoing convergence of voice, broadcast, and data networks leads to a demand for a novel flexible and high-performance packet-forwarding technology. Flexibility is needed for short development cycles and for the support of evolving protocols and standards, combined with the shift towards high-performance packet handling [25] due to the increasing bandwidth demands. Today, packet handling is performed by application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), or general-purpose processors (GPPs). While ASICs have clear advantages in terms of performance, the hardware functions do not provide sufficient flexibility. In contrast, packet forwarding based on GPPs provides high flexibility, but insufficient performance because GPPs were not designed with packet forwarding in mind. Finally, FPGAs can be reprogrammed at gate level, combining features from ASICs and GPPs [18]. However, high-level programmability of FPGAs still is very limited despite improvements of the level of abstraction [5].

Network processors (NPs) are specifically designed processors for combined fast *and* flexible packet handling [11]. Typically pairing an embedded processor complex with application-specific instructions and coprocessor support, NPs can

^{*} An earlier, shorter version of this paper was presented at ANTA 2002 [15]

achieve even higher-layer packet processing at line speeds of several Gb/s. Besides the instruction set, the entire design focuses on high-performance packet processing, including memory, hardware accelerators, bus, and I/O architecture. NPs are well suited for most packet-processing tasks, ranging from content switching, load balancing, traffic conditioning, network security, and terminal mobility to active networks [4, 11]. This paper focuses on the specific role of NPs in ANs.

Recent work has resulted in secure and manageable AN approaches [3, 6, 20]. This paper argues that the remaining performance concerns can be addressed with NPs because their balance between hardware and software efficiently addresses the demand for high data-path performance without sacrificing programmability.

The remainder of the paper is structured as follows. The next section introduces NP architectures and their potential applications. The general benefits of NPs for implementing ANs are described in Section 3. The specific advantages of a concrete AN framework are shown in Section 4, and our experience from its implementation is presented in Section 5.

2 Network Processor Architectures

The main goal of NPs is to provide high processing power and fast packet-oriented I/O, the latter being one of the key GPP bottlenecks. As a typical result, on-chip memory is small due to chip area constraints, arithmetic units are much simpler compared to GPPs and high speed is achieved through parallel packet processing by multiple threads in order to hide memory latencies. In addition, common packet-handling functions, such as prefix matching, classification, metering, policing, checksum computation, interrupt and timer handling, bit manipulation, and packet scheduling, are frequently supported by dedicated hardware units. Moreover, NPs are often teamed with an embedded GPP for more complex tasks [4].

Depending on the location in the network (i.e., edge or core), NP architectures differ in terms of hardware design. The bus architecture as well as the size and type of memory units vary considerably. Edge-type NPs are better equipped for intelligent and stateful packet processing, whereas core-type NPs focus on processing aggregated traffic flows rather than individual packets.

A main NP design decision is whether packet processing prefers the *run-to-completion* or the *pipeline* model [4, 8, 11]. The former dedicates a single thread to packet forwarding from the input interface to the router's packet switch and, likewise, from the switch to the output interface. Threads run on a fixed number of core processors, sharing the memory units, such as lookup trees, instruction memory, and global packet buffers. An alternative to run-to-completion is the pipeline model. Here, the forwarding process is divided into different stages, and each stage is handled by another core processor with its own instruction memory [24].

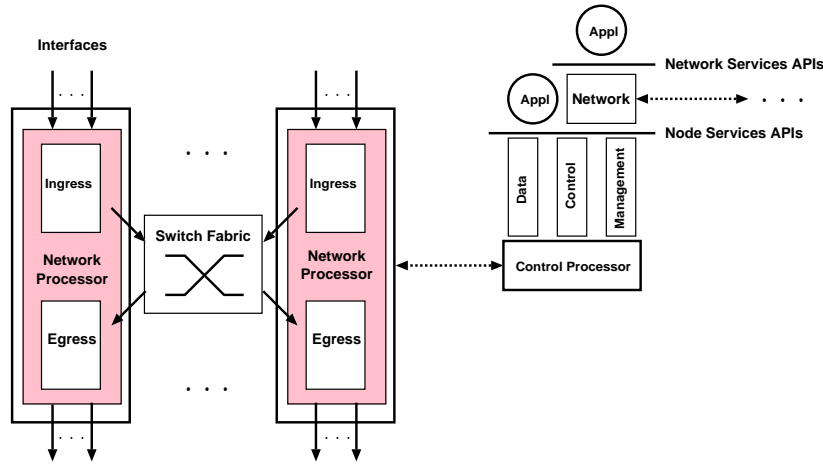


Fig. 1. Network processor programmability.

2.1 Programmability

The NP hardware is typically combined with a horizontally layered software architecture (see Figure 1). On the lowest layer, the NP instruction set provides means for direct packet handling. Many NPs allow access to the instruction memory from a separate control processor, enabling extension or update of the router functionality at runtime.

On the control processor, application programming interfaces (APIs) provide an abstract view of the resources of the network node [1]. The APIs are dedicated to different data-plane, control-plane, and management-plane services of the NP (e.g., initialization, interface configuration, memory and address management, forwarding and address resolution, traffic engineering, classification, diagnostic, event logging, debugging) and can be used for developing portable network applications [4, 9, 11].

2.2 Applications

The use of NPs is beneficial in network applications and services which depend on high data rates as well as rapid development and deployment. In particular, the following application domains have shown benefits from using NPs:

Content switching and load balancing. Information access via HTTP, FTP, or multimedia streaming protocols can result in heavy load on the server. Content switching and load balancing address this problem by transparently distributing client requests across different servers [10, 14].

Traffic differentiation. Quality of service (QoS) and traffic engineering approaches in IP networks require traffic differentiation based on classification,

conditioning, and forwarding functions at edge and core routers. These functions increase data-plane processing and are likely to continue evolving in the future, requiring the flexibility provided by NPs.

Network security. With the increase in manual and automated attacks, security functions are needed for protection of systems and network, such as encryption, intrusion detection, and firewalling. This again increases the need for flexible and evolving data-plane processing, a prime opportunity for NPs.

Terminal mobility. The protocols used in mobile/IP convergence networks [2,22] are likely to evolve in the near future. NPs help mobile-equipment manufactures to adjust their products to the latest standards much faster than with dedicated hardware-based solutions. An alternative GPP/software-based solution would not be able to sustain the necessary stateful higher-layer packet processing at wire speed [11].

Active networking. In AN, packets are no longer passively forwarded, but code carried in packets can actively influence the forwarding process at routers. They require not only significantly more data-plane processing, but can only be implemented if routers expose their state of operation and allow the reconfiguration of forwarding functions. Several of the above domains could also benefit from AN-based implementations.

The rest of the paper will focus on the relationship between ANs and NPs.

3 General Advantages of NP-based ANs

The key idea of ANs [23] is to decouple network services from the networking infrastructure. This is achieved with *active packets* and *active nodes*. The execution environment (EE) on active nodes is typically set up to interpret active programs in byte-code for security and platform independence.

Unfortunately, the interpretation of byte-coded active programs significantly increases the processing overhead per packet. This demand for more performance cannot be addressed with hardware-based forwarding solutions. Routers implemented in ASIC or with FPGAs cannot provide the level of programmability required for active nodes. In the remainder of this section, we discuss how some typical AN concepts can benefit in terms of performance and/or ease of development when implemented on an NP.

Active programs get to the router in two basic forms: as *capsules* and *plugins*. Capsules are self-contained packets including active code that are interpreted or just-in-time compiled [16]. The language used, such as SNAP [19], should be safe with respect to network-resource usage and evaluation isolation. Plugins [13] differ from capsules as the packets contain URL-like pointers, whose contents are then retrieved (if not already cached) and executed by the EE. Both approaches can only gain from NPs: The code can be executed directly on the line card at high speed and direct access to all allowed components.

3.1 Application-Level Multimedia Filtering

In unicast scenarios with peer-to-peer communication between end users, it is possible to negotiate or sense optimum sending rates. However, in multicast scenarios, network resources are difficult to use effectively because the bandwidth up to a congested router may be partially wasted. Positioning application-level packet filters in multicast trees (e.g., preferential dropping of MPEG B-frames) can result in a much better overall link utilization while preserving quality.

It has been proposed that ANs perform application-level multimedia filtering [7], where filters are injected into a network so that an optimum reservation tree is created. The hardware classifiers provided by NPs as well as the corresponding classifier APIs at the control-processor level would make it very easy to implement application-level multimedia filtering.

3.2 Network Management

Active networking for network management [12, 17, 21, 26] results in significantly fewer implementation problems because only few active packets are injected into a network. In general, the forwarding of network-management packets is not time-critical as monitoring and configuration tasks operate on a larger time scale than control- or data-plane tasks. NPs typically provide mechanisms to direct such non-performance-critical packets to the control processor (e.g., using IP header options).

The control processor is equipped with APIs to support typical network-management operations, such as querying and configuring the network nodes. However, active packets sent out for network management-purposes are not used to set and obtain node parameters only, but may include in the code they execute some intelligence for preprocessing and aggregating information from several managed nodes before sending it back to the management station. Such a distributed approach to network management can prevent management stations from becoming overwhelmed with messages, and ensures that the load incurred due to network-management traffic remains low.

4 Advantages of our NP-based AN Framework

The advantages of NPs for implementing and deploying ANs have been described in general terms in the previous section. This section introduces a flexible and generic framework for active networking that matches the functionality provided by NPs in order to exemplify the power of the NP/AN relationship.

The goal of this framework is to enable new networking functionality (i.e., focused on QoS support) which can be dynamically deployed while maintaining architecture neutrality. Therefore our framework relies on a capsule-based approach providing flexible and fast execution of active packets in the data path but also allows active code to be stored on active nodes. Most programming languages are unpredictable in terms of resource consumption and therefore inappropriate for safe active networking. A suitable active-networking programming

language needs to trade off functionality for flexibility while taking into account security. From the considerable number of security issues entailed by ANs we derive the following requirements such an approach has to comply with.

Safe byte-code language. Architectural neutrality, intrinsic safety properties (intrinsic bounds on CPU, memory, and networking bandwidth), and applicability of the language to current and future application domains are prime criteria when designing or choosing the language.

Resource bound. Resources need to be bounded along two axis: per-node resources and the number of nodes/links the packet will visit.

Safety levels: An appropriate safety hierarchy monitors control-plane and data-plane activities. The handling of active-networking packets is divided into six safety levels as is shown in Table 1. The means for data-plane safety are given through the byte-code language. Safety levels 3–5, called the higher safety levels (HSLs), require admission control at the edge of network domains (i.e., the edge of the domain of trust) using policies depending on the network needs. This also enables easy accounting and charging for active packets. Adding and removing dynamic router services requires a public-key infrastructure for integrity and authentication of active code. Alternatively, higher-level packets can be filtered or, better, disabled for a certain domain only.

Sandbox environment: Any active byte-code is executed in a safe environment called the active networking sandbox (ANSB). Information exchange with the router is protected by so-called router services.

Router services: Router services dynamically enhance router functionality to overcome limitations of the byte-code instructions. They can be static, i.e., defined as opcodes in the byte-code language (e.g., IP address lookup, interface enumeration, flow queue management, or congestion status information), or dynamic (e.g., installation of active code into the ANSB for active queue management (AQM) or scheduling, policy manipulation using a dynamically loaded router service). Dynamic router services are usually tailored to networking tasks with a focus on control-plane functionality, and take significantly more time to execute than normal byte-code instructions do. Therefore, router services belong to the set of instructions with a safety level higher than 1. The installation of new router services is restricted to safety level 5. Such an active packet contains the context for which the new service is applicable, and the code section to be installed is given in the packet’s payload.

Routing: Active packets will not interfere with routing protocols. Alternative routes can be proposed by router services as long as the corresponding entries are defined in the local routing table.

In general, we distinguish between safety and security. Safety is given through the definition of the byte-code language itself, the safety hierarchy, and the safe EE for active code. The goal of safety is to reduce risks to the level of traditional IP networks. Security can only be provided by additional network security

Table 1. Safety hierarchy in active networks.

Safety level	Allowed network functionality	Packet and router requirements
5	Dynamic router services (active code): registering new router services	Authentication of active packets needed using a public-key infrastructure.
4	Complex policy insertion and manipulation	Admission control at the edge of network domains; trusted within a domain.
3	Simple policy modification and manipulation	Running in a sandbox environment, limited by predefined rules and installed router services.
2	Creation of new packets and resource-intensive router services (lookups etc.)	Sandbox environment based on the knowledge of the instruction performance.
1	Simple packet byte-code	Safety issues solved by restrictions in the language definition and the use of a sandbox.
0	No active code present in packets	Corresponds to traditional packet-forwarding process.

services including cryptography, authentication, and integrity mechanisms, used to protect the code executed at higher safety levels. This combination achieves both fast packet forwarding in the data path and secure and programmable control-path.

5 Implementation Experience

5.1 Offloading of AN Functionality

The traditional NP control point (NPCP) does not necessarily run on the same GPP as the ANSB and that it even makes sense to separate or dynamically offload AN functionality. For example, the NPCP can run on the external GPP while the higher safety levels of the ANSB are offloaded to the embedded PowerPC (ePPC) available on the PowerNP. The ANSB obtains resources and behavior bounds¹ assigned by the NPCP and administrates them autonomously. Given the distributed layout, which enhances the robustness of the architecture, this is certainly the preferred model. Figure 2 gives an overview of the model based on an IBM PowerNP 4GS3 network processor. In the current implementation, both the NPCP and the ePPC run a Linux 2.4.17 kernel.

In contrast with a standard Linux router without NP, routing and MAC information maintained in the Linux kernel are automatically mirrored to the NP

¹ The behavior bound consists of a classifier describing to whom the service will be offered, a traffic specification (e.g., sender Tspec), and a resource bound vector that characterizes the maximum resource usage of the router service.

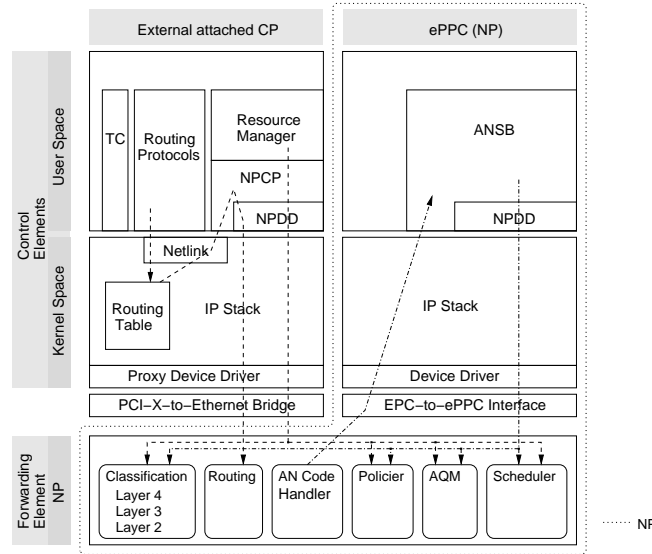


Fig. 2. Architectural overview of the implementation when the ANSB is offloaded to the ePPC.

by the NPCP task, enabling the direct use of many standard control-plane applications. NPCP uses the NP APIs provided by the NP device driver (NPDD). These APIs are also used by NP-aware applications, e.g., a resource manager setting up QoS parameters (e.g., Diffserv over MPLS, flow control).

The AN part is separated from the NPCP as follows. Safety levels 0 and 1 are handled by the active-networking code handler in the data path of the NP. All higher safety levels are offloaded to the ANSB on the ePPC. The ANSB then effectuates NPDD API calls for configuring the NP within the limits of configured policies attributed to the ANSB.

5.2 Packet Definition

Similar to Smart Packets [21] our approach sits directly on top of the networking layer, utilizes the router alert IP header option to indicate active packets, and inserts an active header and code between the IP header and payload. Our approach is a dialect of the SNAP active networking language [19] which allows limited backward loops while still maintaining the safety properties [16]. This approach is downward-compatible, as SNAP-unaware routers will just treat the packet according to safety level 0 and forward them as normal IP packets.

The SNAP header holds information on the safety levels of the active packet in two fields. The first is the initially assigned safety level (IASL) and contains the safety levels in which the packet operates according to the creator of the packet.

The second holds the domain-specific safety levels (DSSL) representing the safety levels applicable in the current domain. They are set by packet classification at the ingress of a domain, and remain valid for the entire path through the domain. This mechanism allows a temporary reduction of the safety level within a given domain.

The NP-based router architecture, which is no longer centralized as NPs can reside on each blade, divides packet processing into two stages: Ingress processing directs packets from the physical interface to the switch, and egress processing does the reverse. Forwarding and classification decisions are usually taken on the ingress, whereas the egress is mainly involved in packet scheduling. This implies that the processing of an active packet can be performed on the ingress as well as the egress side. Consequently, two entry points have to be maintained.

To minimize data swapping during active packet execution, the memory section of the packet is situated between the packet header and the active-code section. In our case the memory section has a maximum size of 128 bytes. The packet payload delivered to an application remains at the end of the packet and can also be seen like a ROM by the active code. By moving the heap and stack into one memory section that is being fixed when the packet is built, more complex error handling (stack overflow) arises but achieves significant improvements that speed up the execution of active packets in an NP.

For deep packet processing, NPs usually handle packet data in blocks of 64 bytes (pages). Hence, branch decisions in data-path active code encounter an additional penalty if the branch target does not lie in the current page. Forward branches require the chained list of pages to be traversed because the location of pages that have not yet been loaded is unknown. Backward branching can load the correct page immediately, as a page history is maintained.

5.3 Data-Path Elements

This section discusses the integration of the active code into the existing data-path processing. The functional behavior of the forwarding code is shown in Figure 3 for the ingress and in Figure 4 for the egress part.

As soon as active packets have been correctly identified in the layer-3 forwarding code (cf. Figure 3) their processing continues in the active-networking code handler. Depending on their functionality, they still might traverse layer-4 processing later. This is the case for HSL-active packets, which require layer-4 classification at domain ingress nodes. The egress part is much simpler as there is no layer-4 classification, and active packet processing can immediately start at dispatch time. HSL-active packets have to be classified already on the ingress side (result is kept in the DSSL field) to avoid unnecessary redirection to the ingress. AQM can be provided on the ingress and/or egress by flow control mechanisms which provide congestion feedbacks signals (i.e., packet arrival rates and queue lengths). Note that there is no layer-4 processing at the egress as layer-4 classification has already been performed on the ingress side.

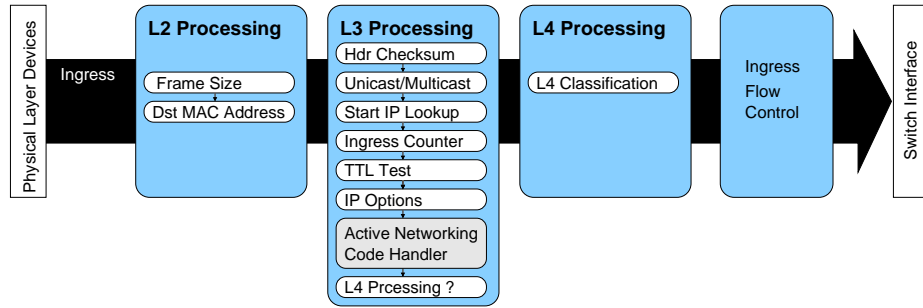


Fig. 3. Ingress data-path processing on a network processor.

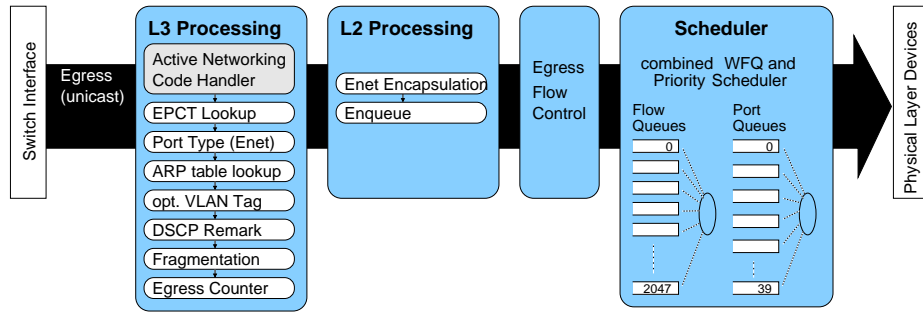


Fig. 4. Egress data-path processing on a network processor.

5.4 Control-Path Elements

HSL-active packets can fulfill control-path tasks. These packets require layer-4 classification and verification of the IASL and DSSL done by the active-networking code handler. Matching packets are then redirected to the ANSB on the ePPC. As can be seen in Figure 3, classification takes place only at ingress and redirection is initiated from there. Possible actions are the deposition of active code (safety level 5) and classifier updates (safety levels 3 and 4) within the behavior bounds. Finally, the ANSB translates updated information (e.g., classifier) into NPDD API calls to reconfigure the NP accordingly. Tasks such as routing and interface management are still maintained by the traditional CP as shown in Figure 2.

6 Conclusion

Despite evident advantages of active-networking technology, ANs still lack the support in mainstream networking products. Many vendors fear that the safety and performance of their platforms will be compromised while other vendors using ASICs are prevented from implementing the flexibility required for ANs. Network processors fill the gap by enabling high performance *and* flexibility.

The paper shows in general and in the context of a specific AN framework that the implementation and deployment of ANs can benefit from network processor technology. The advantages are linked to improved performance and simplified development.

The specific NP framework for demonstrating the beneficial AN/NP relationship allows to tap the power of ANs without sacrificing the safety of traditional IP networking. The main security and safety advantages result from the combination of a byte-code language with intrinsic safety properties, a lean 6-level safety hierarchy enabling control-plane functionalities and persistent active code in active nodes, a sandbox environment for code execution, and off-loading of active-networking functionality from the control point to the NP's GPP processor. This isolation provides a physical barrier in our implementation between the packet-processing core of the NP (i.e., the embedded processor complex), the ePPC running the active networking sandbox, and the control and management functions provided by the control point GPP. We believe that this approach will lead to a wider acceptance of AN in networking devices.

References

1. Network Processing Forum. <http://www.npforum.org/>.
2. The 3rd Generation Partnership Project (3GPP). <http://www.3gpp.org>.
3. 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 and Networks*, 3(1):4–18, Mar. 2001.
4. J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Freléchoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel, and M. Waldvogel. IBM PowerNP network processor: Hardware software and applications. *IBM Journal of Research and Development*, 47(2/3):177–194, Mar./May 2003.
5. F. Braun, J. Lockwood, and M. Waldvogel. Protocol wrappers for layered network packet processing in reconfigurable networks. *IEEE Micro*, 22(1):66–74, Jan./Feb. 2002.
6. M. Brunner, B. Plattner, and R. Stadler. Service creation and management in active telecom networks. *Communications of the ACM*, 44(4):55–61, Apr. 2001.
7. I. Busse, S. Covaci, and A. Leichsenring. Autonomy and decentralization in active networks: A case study for mobile agents. In *Proceedings of International Working Conference on Active Networks*, pages 165–179, 1999.
8. P. Crowley, M. E. Fiuczynski, J.-L. Baer, and B. N. Bershad. Characterizing processor architectures for programmable network interfaces. In *Proceedings of the ACM International Conference on Supercomputing*, pages 54–65, May 8–11, 2000.

9. S. Denazis, K. Miki, J. Vicente, and A. Campbell. Designing interfaces for open programmable routers. In *Proceedings of International Working Conference on Active Networks*, pages 13–24, July 1999.
10. Z. Genova and K. Christensen. Challenges in URL switching for implementing globally distributed Web sites. In *Proceedings of the Workshop on Scalable Web Services*, pages 89–94, Aug. 2000.
11. R. Haas, C. Jeffries, L. Kencl, A. Kind, B. Metzler, R. Pletka, M. Waldvogel, L. Freléchoux, and P. Droz. Creating advanced functions on network processors: Experience and perspectives. *IEEE Network*, 17(4), July 2003.
12. A. W. Jackson, J. P. G. Sterbenz, Condell, and R. Hain. Active network monitoring and control: The SENCOMM architecture and implementation. In *Proceedings of DARPA Active Networks Conference and Exposition (DANCE)*, pages 379–393, May 2002.
13. R. Keller, L. Ruf, A. Guindehi, and B. Plattner. PromethOS: A dynamically extensible router architecture supporting explicit routing. In *Proceedings of Int. Working Conf. on Active Networks IWAN*, pages 20–31, Dec. 2002.
14. L. Kencl and J.-Y. L. Boudec. Adaptive load sharing for network processors. In *Proceedings of INFOCOM '02*, June 2002.
15. A. Kind. The role of network processors in active networks. In *Proceedings of the First International Workshop on Active Network Technologies and Applications (ANTA 2002)*, Tokyo, Japan, Mar. 2002.
16. A. Kind, R. Pletka, and B. Stiller. The potential of just-in-time compilation in active networks. *Proceedings of IEEE OPENARCH '02*, June 2002.
17. J. Kornblum, D. Raz, and Y. Shavitt. The active process interaction with its environment. *Computer Networks*, 36(1):21–34, Oct. 2001.
18. J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor. Reprogrammable network packet processing on the field programmable port extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays (FPGA '01)*, pages 87–93, Feb. 2001.
19. J. T. Moore, M. Hicks, and S. Nettles. Practical programmable packets. In *Proceedings of INFOCOM '01*, Apr. 2001.
20. S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee. Strong security for active networks. *Proceedings of IEEE OPENARCH '01*, pages 63–70, Apr. 2001.
21. B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart Packets: Applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1):67–88, Feb. 2000.
22. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960, IETF, Oct. 2000.
23. D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *ACM Computer Communication Review*, 26(2):5–18, Apr. 1996.
24. M. Venkatachalam, P. Chandra, and R. Yavatkar. A highly flexible, distributed multiprocessor architecture for network processing. *Computer Networks*, 41(5):563–586, Apr. 2003.
25. T. Wolf and J. Turner. Design issues for high-performance active routers. *IEEE Selected Areas in Communications*, 19(3):404–409, Mar. 2001.
26. Y. Yemini, A. V. Konstantinou, and D. Florissi. NESTOR: An architecture for self-management and organization. *IEEE Journal on Selected Areas in Communications*, 18(5), May 2000.