RZ 3471 (# 93960) 01/06/03
Computer Science 6 pages

Research Report

Andreas Kind

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
ank@zurich.ibm.com

The Role of Network Processors in Active Networks

Andreas Kind
IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
ank@zurich.ibm.com

Abstract

This paper argues that network processors can play an important role in the implementation and the deployment of active networks. The balance between hardware and software supplied with network processors addresses the demand for performance and programmability in active networks. The paper introduces network processors and discusses their relationship with active networks in general. Furthermore, a specific example of an advanced active networking function is given that demonstrates the capabilities of active networks based on network processors. The implementation is described for the IBM PowerNP network processor.

1 Introduction

The current convergence of voice and data networks leads to a demand for a new flexible and high-performance packet forwarding technology. Flexibility is needed for short development cycles and for the support of evolving protocols and standards. The demand for high-performance packet handling arises from the tremendous increase of connected end-users and end-devices using a new generation of network services and applications. The technology typically used for packet handling today is either based on 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 for short development cycles. In contrast, packet forwarding based on GPPs provides high flexibility, but insufficient performance because they are not specifically designed for packet forwarding. Finally, FPGAs can be reprogrammed at gate level, resulting in flexibility and performance that is between what can be achieved with ASIC-based and GPP-based solutions. However, the level of programmability of FPGAs still is very limited.

A network processor is a processor that is specifically designed for fast *and* flexible packet handling. It is typically based on an embedded processor complex for parallel execution of an application-specific instruction set. With the support of co-processors, even higher-layer packet processing can be achieved at multiple Gb/s line speeds. Not only the instruction set is customized for packet processing and forwarding, but also the entire design of the network processor, including memory, hardware accelerators, and bus architecture, is targeted to high-performance packet handling.

There are a number of promising application domains for network processors. Among them are content switching, load balancing, traffic conditioning, network security, terminal mobility, and active networks. This paper focuses on the specific role of network processors in the application domain of active networks.

Active networking is an approach to network architecture in which network nodes perform customized computations on packets being forwarded [1]. On the one hand, active networking provides opportunities for new distributed network services and enables dynamic network infrastructure innovation. On the other hand, performance, security, and manageability problems have made it difficult to apply active networking in production networks. Recent work on security and management in active networks [2]–[4] has resulted in secure and manageable approaches to active networking. This paper argues that the remaining performance concerns can be addressed with network processors because their balance between hardware and software addresses the demand for high data path performance while not sacrifying programmability.

The remainder of the paper is structured as follows. The next section introduces network processor architectures. The relation between network processors and active networks is described in Section 3. An implementation of a just-in-time compiler for the IBM PowerNP network processor is briefly presented in Section 4. This implementation of an advanced active networking function highlights the capabilities of network processors in active networks.

2 Network Processor Architectures

Network processors are designed to meet the requirements for performance and programmability of packet forwarding in future IP routers. Compared with GPPs, network processors have, for instance, much simpler arithmetic and caching units, but support parallel packet processing with multiple threads of execution instead. In addition, common packet handling functions, such as tree lookup, classification, metering, policing, checksum computation, interrupt and timer handling, bit manipulation, and packet scheduling, are supported with co-processors or specific functional hardware units.

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

Moreover, a main design decision with network processors is whether packet processing is based on a *run-to-completion* or *pipeline* model [5, 6]. In the run-to-completion model a single thread is dedicated to the packet forwarding process from the input interface to the switch interface and, likewise, from the switch interface to the output interface. Threads run on a fixed number of core processors, which share the memory units, such as lookup trees, instruction memory, and global packet buffers. An alternative to the run-to-completion model is the pipeline model. In this model the forwarding process is divided into different stages, and each stage is handled by a different core processor.

2.1 Programmability

The network processor hardware is typically combined with a horizontally layered software architecture (see Figure 1). On the lowest layer, the network processor instruction set provides direct means for handling incoming packets. Development tools (e.g., compiler and debugger) support a standard software development approach for programming a network processor already on this lowest layer. Some network processors allow access to the instruction memory from a separated control processor, so that the standard forwarding behavior written in the network processor instruction set can be extended or updated even during operation of a router. The control processor is typically connected via an inter-process communication protocol to the network processor. Control-plane packets can then be exchanged between the network processor and the control processor.

On the control processor, Application Programming Interfaces (APIs) provide an abstract view onto the resources of the network node. The APIs are dedicated to different data-plane, control-plane, and management-plane services of the network processor (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 [7, 8, 6]. Together with another software layer spanning over more than one network node, for instance using protocols from policy-based networking, network-wide services can be implemented in a simple and fast way.

2.2 Network Processor Applications

There are many network applications and services that depend on high data rates as well as rapid development and deployment. These are likely to provide opportunities for using network processors. In particular, the following application domains may benefit from the network processor approach:

Content switching and load balancing. The client-server model of information access via HTTP or FTP is widespread within the Internet. For popular information sites, a server may have to handle the requests of numerous clients with reasonable response times. Such situations can lead to server overload and network congestion close to the server. Content switching and load balancing address this problem by transparently distributing client requests to different server machines [9, 10].

Traffic differentiation. Quality of Service and traffic engineering approaches in IP networks require traffic differentiation based on classification, conditioning, and forwarding functions at edge and core routers [11, 12]. These functions increase data-plane processing and are likely to continue evolving in the future, so that an implementation using network processors is sensible.

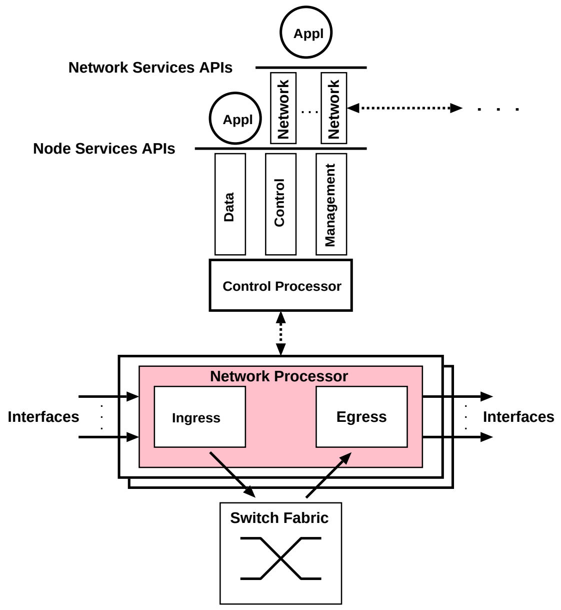

Figure 1: Network processor programmability.

Network security. Because business and public institutions increasingly make use of the Internet, security functions, such as encryption, intrusion detection, and firewalling, are needed. The resulting increase in data-plane processing with security functions provides opportunities for network processors also in this case.

Terminal mobility. The convergence of mobile and IP networks requires edge routers to support new network functions, such as tunneling [13] and bundling [14] of data streams between wireless end terminals and IP backbones. Appropriate protocols are currently being standardized and are likely to evolve over the next few years. Network processors can help mobile equipment manufactures to adjust their products to the latest standards much quicker than with dedicated hardware-based solutions. A software-based solution, as the other alternative, would not be able to support wire-speed forwarding in conjunction with stateful packet processing and high-layer packet parsing.

Active networking. In active networks packets are no longer passively forwarded, but code carried in packets can actively influence the forwarding process at routers. Active networks require not only significantly more data-plane processing, but can only be implemented if routers expose their state of operation and allow reconfiguration of forwarding function.

The focus in the remainder of the paper is on the relation between active networks and network processors.

3 Active Networks and Network Processors

The key idea of active networks [15] is to decouple network services from the underlying networking infrastructure. This is achieved with *active packets* and *active nodes.* Active packets are extended data packets that are sent either from end-user applications, active network gateways, or network management applications through an active network domain. They carry code for execution at traversed active nodes either directly or by reference. If the code is carried by reference, a separate code distribution mechanism needs to be in place.

Active nodes provide an execution environment for running active code. The execution environment controls the access to node resources (e.g., link state, routing table, and congestion status) and enables the creation of new active packets as well as the modification of the active packet being currently handled. In some systems active packets can pick up or leave soft state. Typically, the execution environment is implemented as a virtual machine that interprets active programs represented in byte-code. This approach provides architectural neutrality and security because the virtual instruction set abstracts from the underlying proprietary router, and the execution of active code can be controlled for security reasons as needed.

Unfortunately, the interpretation of byte-coded active programs significantly increases the processing overhead per packet. This demand for performance can, however, not be addressed with hardware-based forwarding solutions. Routers implemented in ASIC or with FPGAs cannot provide the level of programmability that is required for an active node.

In the remainder of this section we discuss how some typical active network applications can benefit in terms of performance and/or ease of development when implemented on a network processor.

3.1 Plug-in Approach

As mentioned, active code does not necessarily have to be inserted in active packets, as instead a reference (similar to a URL) can be carried in the packet. This approach is promising if code locality at routers is to be expected. Once an active code component has been downloaded and installed, it does not have to be retrieved again when another packet at the same router refers to it.

When this plug-in approach is seen from a network processor point of view, two options for the installation of the active code component exist. If the component contains a performance-critical code segment with the purpose of extending the forwarding code, it has to be dynamically linked into the already existing forwarding code. This may be as interpreted byte-code for a preinstalled virtual machine or as native core processor code. In the former case, the active code is stored in the data memory; in the latter case the active code is stored in the instruction memory.

If the downloaded component is dedicated to network control or management, it can be linked into an execution environment at the control processor. In this case, the APIs at the control processor can greatly simplify the mapping of the functions that can be used by active programs onto the functions that are actually available on the network processor. For instance, a Differentiated Services [11] plug-in component can directly make use of the services APIs dealing with dynamic creation and deletion of classifier rules, traffic markers, and traffic shapers.

3.2 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., dropping only MPEG B-frames) can result in a much better overall link utilization.

It has been proposed that active networks perform application-level multimedia filtering [16]. In this approach, filters are injected into a network so that an optimal reservation tree is created. The hardware classifiers provided with network processors as well as the corresponding classifier APIs at the control processor level would make it very easy to implement application-level multimedia filtering.

3.3 Network Management

Active networking for network management [17] 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. Network processors 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 solely used to set and get node parameters, 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 by network management traffic remains low.

4 Example: Just-in-Time Compilation

Poor execution speed of active programs represented in byte-code is mainly due to the interpretation overhead of the virtual machine. With just-in-time (JIT) compilation, a byte-coded active program about to be executed for the first time, is compiled into native network processor instructions and then executed as native binary [18].

This technique is an advanced active networking function that uses the available processing headroom (i.e., the forwarding resources that are not already allocated for standard packet processing) for compilation and native execution of an active network program, rather than for its interpretation. An active network language is assumed that allows static analysis to find out whether the binary execution is safe.

The cost of compiling active programs is the key factor that determines whether JIT compilation is feasible for active networks at all. Once a program is compiled, its execution will be an order of magnitude faster than interpretation. A performance benefit is achieved if the interpretation time I exceeds the sum of compilation time C and native execution time E, i.e., $I > C + E$.

Clearly, implementing an active network JIT compiler on a network processor poses some considerable challenges:

Stack-based *vs.* register-based execution. Because network processors in general are register-based, a JIT compiler for active networks has to translate the, typically, *stack-oriented* operation of byte-code instructions to the *register-oriented* operation of native network processor instructions. This requirement does not necessarily increase the complexity of the compiler. As the stack during the execution of an active program cannot grow very large owing to given run-time resource bounds, all stack positions can be mapped directly onto a subset of the available network processor registers. The mapping from relative stack positions to absolute registers can, however, be determined only during compile-time if it is possible to associate a fixed stack depth with each source instruction. In other words, an instruction in a source program must not be executed at different stack levels—as would be the case with true recursion. Allowing only this restricted kind of recursion (also known as tail-recursion) in combination with a maximum stack size limited by the number of available native registers, allows simplifying the JIT compiler while still retaining the possibility of handling `while` and `for` loops. Complex register-allocation schemes, including register flushing into memory, that would complicate the compiler are not necessary.

Compilation time. The source program representation inside active packets is linearized and in byte-code format, so that the compilation process does not include costly compilation phases, such as tokenizing, syntax/semantic analysis. In fact, it can be performed in one pass. The compilation time for unfragmented active programs (smaller than the common MTU size) is, therefore, short, and the cycles spent for JIT compilation per packet are likely to fit into the available data-path headroom.

Access to instruction memory. The JIT compiler has to write the compiled active program into the instruction memory of the network processor before execution can start. Thus, the target network processor for a JIT compiler has to support write access to the instruction memory. Also the free instruction memory has to be sufficiently large to accommodate the compiler itself.

Relative to interpretation, the compilation overhead incurred by JIT compilation is reduced if the compiled code is cached either at nodes (i.e., basically not removed from instruction memory) or inside the packet. In the first case, packets of the same flow could reuse the compiled native code. In the second case, a packet can reuse native code if another router on the way to the destination supports the same native instruction set.

When considering code caching, tail recursion, and loops, a performance benefit is achieved with JIT compilation when $nI > C + nE$. The number of times a native program can be executed without recompilation is given by n. Assuming that interpretation is much more expensive than native execution ($I \gg E$), JIT compilation pays off when $nI > C$.

A JIT compiler as described above has been implemented for the IBM PowerNP network processor [18]. For the PowerNP, we found that compilation is only slightly more expensive than interpretation, so that already with $n \geq 2$ a performance benefit can be expected.

5 Conclusion

The applicability of active network technology in production networks requires high performance *and* programmable routers, which suggests the use of network processors for active networking. The paper supports this idea by showing how active networking applications can benefit from underlying network processor architectures in terms of performance and ease of development.

References

[1] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[2] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee, "Strong security for active networks," *Proceedings of IEEE OPENARCH '01*, pp. 63–70, Apr. 2001.

[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*, vol. 3, no. 1, pp. 4–18, Mar. 2001.

[4] M. Brunner, B. Plattner, and R. Stadler, "Service creation and management in active telecom networks," *Communications of the ACM*, vol. 44, no. 4, pp. 55–61, Apr. 2001.

[5] 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*, May 8–11, 2000, pp. 54–65.

[6] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel, and M. Waldvogel, "PowerNP network processor hardware, software, and applications," *IBM Journal of Research and Development*, 2002, to appear.

[7] J. Biswas, A. Lazar, J. Huard, K. Lim, S. Mahjoub, L. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein, "The IEEE P1520 standards initiative for programmable network interfaces," *IEEE Communications Magazine*, vol. 36, no. 10, pp. 64–70, Oct. 1998.

[8] S. Denazis, K. Miki, J. Vicente, and A. Campbell, "Designing interfaces for open programmable routers," in *Proceedings of International Working Conference on Active Networks*. July 1999, vol. 1653 of *Lecture Notes in Computer Science*, pp. 13–24, Springer, Berlin.

[9] Z. Genova and K. Christensen, "Challenges in URL switching for implementing globally distributed Web sites," in *Proceedings of the Workshop on Scalable Web Services*, Aug. 2000, pp. 89–94.

[10] L. Kencl and J.-Y. Le Boudec, "Adaptive load sharing for network processors," in *Proceedings of INFOCOM '02*, June 2002.

[11] S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, IETF, Dec. 1998.

[12] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, IETF, Jan. 2001.

[13] "The 3rd Generation Partnership Project (3GPP)," http://www.3gpp.org, Mar. 2002.

[14] 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.

[15] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *ACM Computer Communication Review*, vol. 26, no. 2, pp. 5–18, Apr. 1996.

[16] 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*. 1999, vol. 1653 of *Lecture Notes in Computer Science*, pp. 165–179, Springer, Berlin.

[17] B. Schwartz, W. Zhou, A. Jackson, W. Strayer, D. Rockwell, and C. Partridge, "Smart Packets for active networks," *ACM Computer Communications Review*, Jan. 1998.

[18] A. Kind, R. Pletka, and B. Stiller, "The potential of just-in-time compilation in active networks," *Proceedings of IEEE OPENARCH '02*, June 2002.