Design of a Network Service Processing Platform for Data Path Customization *

Qiang Wu and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA, USA
{qwu,wolf}@ecs.umass.edu

ABSTRACT

Custom packet processing functionality in routers is one of the key characteristics of next-generation Internet architectures. Network services have been proposed as an abstraction to describe, compose, and deploy end-to-end connections with custom communication features. We present a novel hardware architecture for high-performance processing of such network services in the data path. The design provides simple processing units to implement services and a custom hardware infrastructure to manage packets and processing context. The design allows for simple software development, flexible network service allocation, and high scalability to handle traffic at Gigabit line rates.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*Routers*; C.1.4 [**Processor Architectures**]: Parallel Architectures

General Terms

Design, Performance

Keywords

Network processor, next-generation Internet, network service

1. INTRODUCTION

Recent research in computer networking has focused on the design of a new Internet architecture [5]. The goal of defining a new architecture is to overcome the limitations of the current Internet in accommodating new protocols and communication paradigms. Unlike the current Internet, where routers are limited to simple packet forwarding [1], next-generation architectures feature routers that support dynamic deployment of novel packet processing functions in the data path. This processing functionality can be seen as a network service that implements custom data path features for each connection [26].

The design of high-performance router systems with flexible network services support is therefore fundamental to next-generation networks. When using general-purpose processors, packet processing functions can be reprogrammed after deployment, but these systems usually consume too much resources (e.g. power, space, etc.) when used for high data rates. For better performance cost ratio, network processors have been developed utilizing multiple embedded processor cores on a single chip. Typical network processors use a variety of memory interfaces, complex interprocessor communications, and hardware accelerators. To develop high-performance software for network processors, an in-depth understanding of the underlying hardware is necessary. However, researchers and practitioners who design and develop novel networking functionality are often not experienced in tuning embedded software for multi-core systems. This dichotomy has limited the deployment of network processor systems in the existing testbeds for next-generation Internet research.

To tackle this problem, we present a novel packet processing platform that provides a very simple processing abstraction to the software developer and yields performance by hiding more complex operational issues (e.g. packet and state management) in hardware. In particular, we use our previously proposed *network services* as processing abstractions for this platform. The design goals of our *network service processing platform* are:

- Separation of complex I/O, inter-processor communication, and state management from packet processing: Packet processing should be performed on a very simple processor model. Hardware support should hide more complex aspects of the system.
- Easy software development: The simplicity of the processor model should allow for easy software development.
- Scalability to large systems: The use of network services as programming abstractions and simple processor models should allow the scaling of the system architecture to support Gigabit data rates.

Guided by these goals, we present a network service processing platform that uses a *service processor* as the basic

*This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-0626690 and CNS-0447873.

PRESTO'09, August 21, 2009, Barcelona, Spain.

processing unit. In this paper, we describe the design and operation of this system. Specifically, our contributions are

- a design of the service processor unit and the overall network service processing platform,
- a description of how network service program instructions and data are managed by hardware to simplify programming of network services, and
- an estimation of the performance of different system configurations.

The remainder of this paper is organized as follows. Section 2 discusses related work. The service processor design is presented in Section 3. The design of the entire network service processing platform is described in Section 4. Section 5 discusses performance estimates. Section 6 summarizes and concludes this paper.

2. RELATED WORK

The concept of programmability in the data path has been introduced in a variety of programmable router designs [14, 18, 27]. Programmable routers are conceptually similar to routers in active networks [23]. The main difference is that programming of an active router is performed in-band, whereas programmable routers are programmed out-of-band. The latter is easier to manage and thus more practical for deployment in an actual network. The packet processing systems of programmable routers are typically implemented using network processors [21, 25], with a number of commercial network processors being available from Intel, AMCC, EZchip, etc. Programmability in network systems has also been proposed on the basis of programmable logic devices [9, 22].

Programming abstractions in network processing systems range from very open environments [7] to very structured environments, where networking functions are represented as modules. Such modules have been proposed for Click [12] and NP-Click [19] as well as for router plugins [3] and hardware plugins [22]. Our network service architecture [6] uses a similarly structured approach, where services usually implement full network functionalities or protocols, which represent self-contained network processing and protocol operations. The concept of network services was first introduced in [26]. Similar approaches permit composition of protocol functionality in form of stacks [8, 15, 24], protocol heaps [2], and per-flow stacks [4]. The runtime management of network processor resources in environments that permit dynamically allocated packet processing functions is discussed in [13, 28, 29]. We leverage these results for the management of service processor resources on the network service processing platform.

3. SERVICE PROCESSOR DESIGN

The overall design of our network service processing platform is based on the idea that we want to perform processing at the granularity of a network service. Thus, the service processor design is tuned to perform the service processing of a packet. Before describing the details of the service processor design, we briefly discuss network services in general.

3.1 Network Services

We define a network service as any type of protocol feature or processing function that occurs within application-to-application data communication. Examples of network services include network address translation, multicast, intrusion detection system, Qos routing [30], functionalities such as SSL termination [11] to provide reliability and privacy, etc. Some network services are confined to a single location (e.g., network address translation); others are used in matching pairs (e.g., reliability, privacy). We envision that the set of available services is globally coordinated (e.g., through IETF standards), and the number of available services is in the order of dozens. When setting up an end-to-end connection, the communicating parties specify which services should be instantiated. The ability to request custom services for each connection provides the necessary flexibility to support new protocols and communication paradigms. Limiting the number of distinct services keeps the complexity of the system manageable, while permitting a nearly unlimited number of service combinations and thus allowing great flexibility.

A connection's service requirements are expressed as a *sequence of services* and can be passed as a parameter to a service socket [20]. The control infrastructure in the network service architecture then places the connection in the network such that service instances are invoked along the path of the connection. We have solved the distributed routing problem of finding the least-cost connection and processing setup in prior work [10]. Once instantiated, we assume routes and service placement remains fixed for the duration of a connection.

Routers in the network service architecture not only forward traffic, but also perform service processing. When receiving a packet, the router needs to determine which (if any) services need to be performed and perform the corresponding processing. Clearly, packets belonging to different connections may require very different service processing. Since some services need to maintain processing state to perform correctly, a router also needs to manage the per-flow and per-service processing state. Our work describes how to implement an efficient network service processing platform that can perform these functions.

3.2 Program Execution on Service Processor

As discussed above, one of the goals of our design is to simplify code development for the network service processing platform. Since network services already provide a clean separation among network processing functionalities, we aim at designing a processing system that can effectively process one packet for one service and expanding it to a larger system.

To achieve the desired simplicity, we use the logical system design and memory layout of a service processor shown Figure 1. The service processor is assumed to be a simple processor core (e.g., ARM-based RISC core) with an interface for reading program instructions and an interface for access to data memory. (If desired, the system can be designed as a von Neumann architecture with unified instruction and data memory. However, code section and data section are usually isolated in network processing and thus the Harvard architecture with separate instruction and data memories shown in Figure 1 can be used.) In the instruction memory, the code for running a particular service is placed at a fixed,

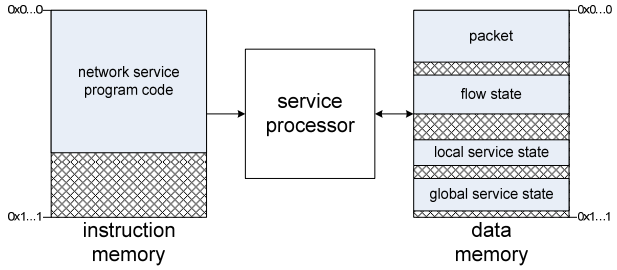

Figure 1: Logical System Design and Memory Layout for Service Processor.

well-known offset (e.g., `0x0`). In data memory, the packet, the flow state, and the local and global service state are also placed at well-known offsets.

With this logical design, packet processing and code development for packet processing becomes straightforward. In order to access packet data, instructions simply need to reference data memory based on the (fixed) packet offset. Similarly, other state information can be accessed. The program code is placed in a fixed location in the instruction memory and thus can be accessed easily by the processor.

Clearly, this simple model cannot be implemented directly in hardware. Multiple services and packets need to be supported by an effective and scalable system. While we describe the implementation for such a system below, it is important to note that the logical view for any given service remains that shown in Figure 1. The network service processing platform ensures that the right instruction memory and data memory context is available whenever a service processor is activated.

3.3 Implementation of Service Processor

To achieve the simple logical perspective in which the service processor core operates, we need to provide a supporting infrastructure to handle physical program and state management. The overall design of the service processor is shown in Figure 2. The processor core is shown in the middle, instruction memory on the left, and data memories on the right.

The main difference to the logical view in Figure 1 is that the instruction memory now contains program code for multiple services and the data memory contains flow and service state for multiple services. Using the address shifter components (explained in more detail below), it is possible to select the program code (and flow and service state) that is accessed by the processor when it operates in its simplified logical view. This multiplexing feature is one of the key aspects of our architecture and provides us with the ability to handle complex program and state management in hardware.

The address shifter component is responsible for mapping the processor core's address space to the memory section that contains the appropriate data. We discuss this functionality in the context of the instruction memory, but it also applies to data memory as shown in Figure 2. In our example, we assume a 16-bit address space used by the processor core. Note that this design can also be parameterized for other memory space sizes. We only present one configuration for illustrative purposes. On the interface between the service processor core and the instruction memory, the least

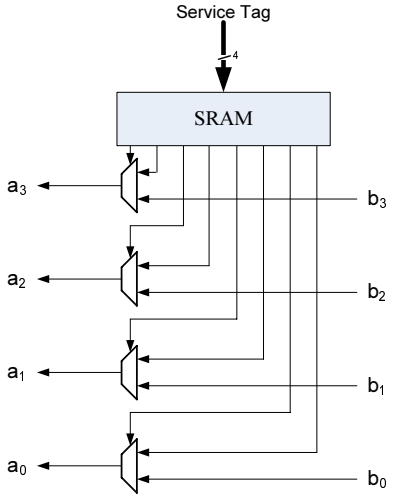

Figure 3: Address Shifter Design.

significant bits (the lower eight bits in the system shown in Figure 2) are connected directly to the instruction memory. Thus, the address translation operates in chunks of 256 bytes. The eight most significant bits are sent into an 8-bit address shifter component. The design of a 4-bit address shifter is shown in Figure 3. By sending the appropriate control signals from SRAM, the address shifter has the ability of either (1) forward an address bit (i.e., outputting b_i on a_i) or (2) overwrite an address bit (i.e., outputting a stored value from SRAM on a_i). With these two options, the service processor core's address space can be shifted to another range in the address space and it can be limited in size (if less than 64kB are needed). For example, the processor uses a 10-bit address space, which is stored in physical memory from `0x0400` to `0x07FF`, address bits $0 \ldots 7$ are connected directly to the memory, address bits $8 \ldots 9$ are forwarded by the address shifter, address bits 10 is set to 1 by the address shifter, and all higher address bits are set to 0. Note that if the overall address space for service programs needs to exceed 16 bits, additional address lines can be controlled by the SRAM in Figure 3 (without having a choice of forwarding processor address lines since they do not exist beyond 16 bits). The selection of signals sent to the multiplexers depends on the *service tag*. The service tag identifies which service needs to be performed by the processor and thus selects the service program that is executed.

3.4 Packet I/O

For the above design to work correctly, we also need to have a mechanism to transfer packets into and out of the service processor. As shown in Figure 2, there is a dedicated packet memory to which part of the logical processor data memory is mapped. For efficient operation, we have designed a novel buffer shown in Figure 4.

The main idea is to have two physical packet buffers available. While one of them used by the processor to access packet memory, the other can be used to transfer in the next packet (after transferring out the previous packet). Once processing of a packet has been completed (signaled by `PKT_DONE`), the buffers are (logically) swapped. The data I/O of service processor is controlled by two FIFOs, which provide data path connections among multiple service pro-

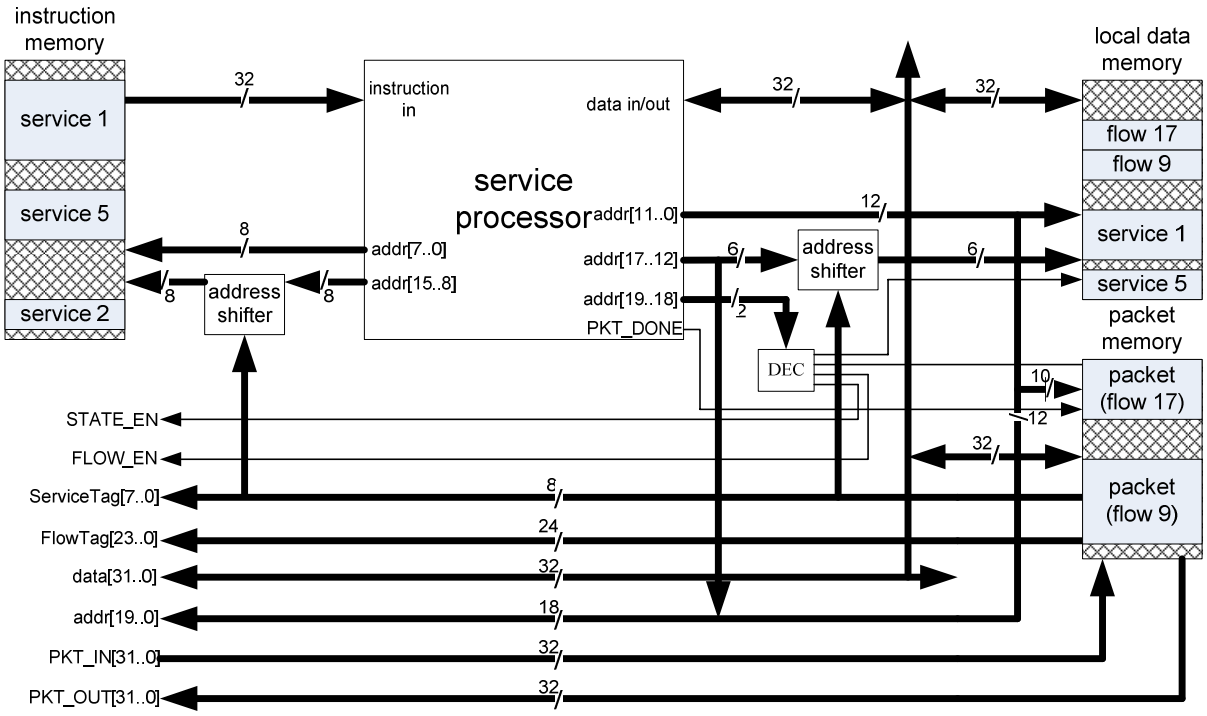

Figure 2: Service Processor Design.

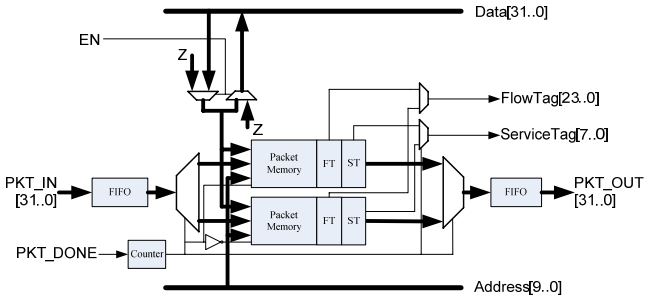

Figure 4: Packet Buffer Design.

cessor units. This design of a packet buffer can be expanded in size to contain more than two packets. Such an expansion may be necessary if moving of packets takes significantly longer than the service processing that is performed.

Each packet also carries meta-information that ensures that it is processed correctly. This information consists of:

- Service Tag: The service tag identifies which service is to be performed by a service processor. As mentioned above, the service tag is used by the address shifter to determine to where to map the core's instruction address space.
- Flow Tag: The flow tag identifies to which flow a packet belongs. This tag is used to demultiplex to the correct flow state information.

These tags need to be set correctly before initiating the processing of a packet. This step is done by the network service processing platform as discussed below. Also, if multiple services are to be performed on a packet, multiple service tags are necessary.

4. NETWORK SERVICE PROCESSING PLATFORM

The overall architecture of the network service processing platform, which uses a number of parallel service processors, is illustrated in Figure 5. In the data plane of the system, packets enter through the I/O interface and get classified into flows. Once the flow has been identified, the system can assign the appropriate service and flow tags to ensure that the correct set of services is applied the packets. Then, packets enter the grid of service processors, where each gets processed as described above. Each service processor has access to shared busses for flow state data and service state data. After completing the processing of the packet, it is schedule for output on the outgoing interface (or the interface connecting to the router switch fabric).

To ensure correct operation, the service information for each flow needs to be set up correctly. This is done via the control plane, where the flow manager and service manage interface with the network service controller. The network service controller performs connection setup, routing, and service placement. Its functionality is described in more detail in [6].

To achieve scalability in the grid of service processor, we do not assume a single shared interconnect for packet transmission between service processors. Instead, we arrange them in a mesh (or possibly multiple parallel pipelines), which requires each service processor only to support one (or a few) local point-to-point connections. Note that the service and flow state bus is still a global interconnect, but we expect it to be utilized at a much lower rate than the packet interconnection.

One of the key operational challenges in this system is the runtime management of resources. In particular, we foresee the following questions:

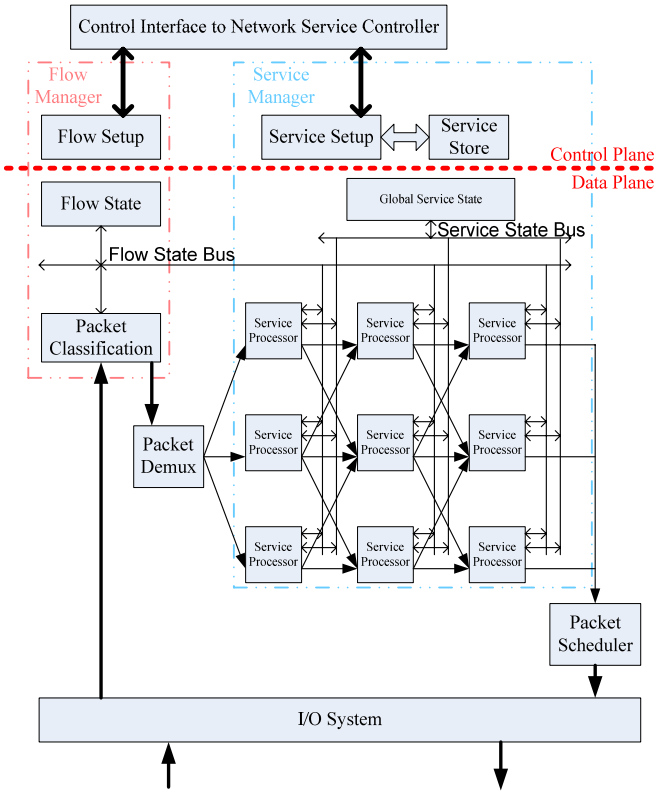

Figure 5: Network Service Processing Platform Design.

- How can flows be mapped to traverse a patch through the grid of service processors while allocating the right set of services to them? Note that it is possible that a packet traverses a service processor without being processed.
- How many (and specifically which) service programs should be installed on any given service processor?
- How can resources be reallocated when requirements change during runtime (e.g., due to an increase of a flow's data rate)?

In our prior work on runtime management of network processors we have already developed a solution to allocating processing resources on demand [29]. This experience will help us in solving the problems specific to the network service processing platform.

5. PERFORMANCE

We provide a rough estimate of the performance of different network services processing platform configurations in Table 1. We assume three configurations with increasing system performance and increasing processing demands. Configuration I assumes a modest 2×2 processor grid with low clock rate and Configuration III represents a high-end system. The cycles per instruction are based on 10ns SRAM access time with an instruction mix containing 10% memory operations (no multi-threading). The effectiveness of the runtime management system to utilize all system resources decreases as more processors are available. The number of instructions executed by a service ranges from 500 to 2,000 per packet [17]. We assume an average packet size 300 bytes. Based on the results in Table 1, we can see that even small configurations of our system can support data rates of several Gigabits per second for simple services. High-end configurations can support tens of Gigabits per second with more complex services.

Table 1: Performance Estimates.

Configuration	I	II	III
Serv. proc. grid size	2×2	4×4	8×8
Serv. proc. clock rate (MHz)	500	1,000	2,000
Cycles per instruction	1.5	2	3
Runtime effectiveness	95%	90%	80%
Effective total MIPS	1,267	7,200	34,133
Service instr. per pkt.	500	1,000	2,000
Packets per second (Mpps)	2.5	7.2	17.1
Data rate (Gbps)	6.1	17.3	41.0

As ongoing research, we are planning to develop an implementation of the network service processing platform on the NetFPGA system [16]. Once this prototype exist, we can evaluate the performance (and chip area requirements) more carefully and compare them to other network processing systems.

6. SUMMARY AND CONCLUSIONS

We argue the programmable packet processing functionality is essential for next-generation networks. To overcome the complexities of current network processor systems, we propose a platform that provides custom data-path process-

ing based on network services. We discuss how our design allows processing context to be managed in hardware to simplify software development. Our initial performance estimates indicate that the system could support Gigabit per second link rates.

7. REFERENCES

[1] Baker, F. Requirements for IP version 4 routers. RFC 1812, Network Working Group, June 1995.

[2] Braden, R., Faber, T., and Handley, M. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Computer Communication Review 33*, 1 (Jan. 2003), 17–22.

[3] Decasper, D., Dittia, Z., Parulkar, G., and Plattner, B. Router Plugins - a modular and extensible software framework for modern high performance integrated services routers. In *Proc. of ACM SIGCOMM 98* (Vancouver, BC, Sept. 1998), pp. 229–240.

[4] Dutta, R., Rouskas, G. N., Baldine, I., Bragg, A., and Stevenson, D. The SILO architecture for services integration, control, and optimization for the future internet. In *Proc. of IEEE International Conference on Communications (ICC)* (Glasgow, Scotland, June 2007), pp. 1899–1904.

[5] Feldmann, A. Internet clean-slate design: what and why? *SIGCOMM Computer Communication Review 37*, 3 (July 2007), 59–64.

[6] Ganapathy, S., and Wolf, T. Design of a network service architecture. In *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Honolulu, HI, Aug. 2007), pp. 754–759.

[7] Goglin, S. D., Hooper, D., Kumar, A., and Yavatkar, R. Advanced software framework, tools, and languages for the IXP family. *Intel Technology Journal 7*, 4 (Nov. 2003), 64–76.

[8] Gu, X., Nahrstedt, K., and Yu, B. SpiderNet: An integrated peer-to-peer service composition framework. In *Proc. of Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (Honolulu, HI, June 2004), pp. 110–119.

[9] Hadzic, I., Marcus, W. S., and Smith, J. M. On-the-fly programmable hardware for networks. In *Proc. of IEEE Globecom 98* (Syndey, Australia, Nov. 1998).

[10] Huang, X., Ganapathy, S., and Wolf, T. A scalable distributed routing protocol for networks with data-path services. In *Proc. of 16th IEEE International Conference on Network Protocols (ICNP)* (Orlando, FL, Oct. 2008).

[11] Khan, E., El-Kharashi, M. W., Ehtesham Rafiq, A., Gebali, F., and Abd-El-Barr, M. Network processors for communication security: a review. In *Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing 2003. PACRIM. 2003 IEEE Pacific Rim Conference onCommunications, Computers and signal Processing (PacRim 2003)* (Waikiki, HI, Feb. 2003).

[12] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The Click modular router. *ACM Transactions on Computer Systems 18*, 3 (Aug. 2000), 263–297.

[13] Kokku, R., Riché, T., Kunze, A., Mudigonda, J., Jason, J., and Vin, H. A case for run-time adaptation in packet processing systems. In *Proc. of the 2nd Workshop on Hot Topics in Networks (HOTNETS-II)* (Cambridge, MA, Nov. 2003).

[14] Kuhns, F., DeHart, J., Kantawala, A., Keller, R., Lockwood, J., Pappu, P., Richard, D., Taylor, D. E., Parwatikar, J., Spitznagel, E., Turner, J., and Wong, K. Design of a high performance dynamically extensible router. In *Proc. of DARPA Active Networks Conference and Exhibition* (San Francisco, CA, May 2002).

[15] Lakshminarayanan, K., Stoica, I., and Wehrle, K. Support for service composition in i3. In *Proc. of the 12th Annual ACM International Conference on Multimedia* (New York, NY, Oct. 2004), pp. 108–111.

[16] Lockwood, J. W., McKeown, N., Watson, G., Gibb, G., Hartke, P., Naous, J., Raghuraman, R., and Luo, J. NetFPGA–an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education* (San Diego, CA, June 2007), pp. 160–161.

[17] Ramaswamy, R., Weng, N., and Wolf, T. Analysis of network processing workloads. In *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (Austin, TX, Mar. 2005), pp. 226–235.

[18] Ruf, L., Keller, R., and Plattner, B. A scalable high-performance router platform supporting dynamic service extensibility on network and host processors. In *Proc. of ACS/IEEE International Conference on Pervasive Services (ICPS'2004)* (Beirut, Lebanon, July 2004).

[19] Shah, N., Plishker, W., Ravindran, K., and Keutzer, K. NP-Click: A productive software development approach for network processors. *IEEE Micro 24*, 5 (Sept. 2004), 45–54.

[20] Shanbhag, S., and Wolf, T. Implementation of end-to-end abstractions in a network service architecture. In *Proc. of Fourth Conference on emerging Networking EXperiments and Technologies (CoNEXT)* (Madrid, Spain, Dec. 2008).

[21] Spalink, T., Karlin, S., Peterson, L., and Gottlieb, Y. Building a robust software-based router using network processors. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)* (Banff, AB, Oct. 2001), pp. 216–229.

[22] Taylor, D. E., Turner, J. S., Lockwood, J. W., and Horta, E. L. Dynamic hardware plugins: Exploiting reconfigurable hardware for high-performance programmable routers. *Computer Networks 38*, 3 (Feb. 2002), 295–310.

[23] Tennenhouse, D. L., and Wetherall, D. J. Towards an active network architecture. *ACM SIGCOMM Computer Communication Review 26*, 2 (Apr. 1996), 5–18.

[24] Vellala, M., Wang, A., Rouskas, G. N., Dutta, R., Baldine, I., and Stevenson, D. A composition algorithm for the SILO cross-layer optimization service architecture. In *Proc. of the Advanced Networks and Telecommunications Systems Conference (ANTS)* (Mumbai, India, Dec. 2007).

[25] Wolf, T. Challenges and applications for network-processor-based programmable routers. In *Proc. of IEEE Sarnoff Symposium* (Princeton, NJ, Mar. 2006).

[26] Wolf, T. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Arlington, VA, Oct. 2006), pp. 79–86.

[27] Wolf, T., and Turner, J. S. Design issues for high performance active routers. In *Proc. of the International Zurich Seminar on Broadband Communications* (Zurich, Switzerland, Feb. 2000), pp. 199–205.

[28] Wolf, T., Weng, N., and Tai, C.-H. Run-time support for multi-core packet processing systems. *IEEE Network 21*, 4 (July 2007), 29–37.

[29] Wu, Q., and Wolf, T. On runtime management in multi-core packet processing systems. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (San Jose, CA, Nov. 2008).

[30] Zhou, W., Lin, C., Li, Y., and Tan, Z. Queue management for QoS provision build on network processor. In *Proc. of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)* (San Juan, PR, May 2003), p. 219.