Support for Dynamic Adaptation in Next Generation Packet Processing Systems

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

***Abstract*—Designs of next-generation Internet architectures propose a diverse and changing set of features in the data path of routers. These routers require high-performance programmable packet processing platforms to allow for dynamic feature deployment and adaptation. In this paper, we present programming abstractions that quantifies processing and storage of network services. We further devise a runtime environment that provides the ability of adaptation to dynamically deployed network services and varying network traffic.**

I. Introduction

In recent years, additions to the data path feature set of routers (i.e., firewall, intrusion detection, network address translation, etc.) has brought the need for flexibility in design of processing nodes inside networks. New services in data path vary greatly in terms of hardware resource requirement and quality of service. And the trend of adopting more services on intermediate processing nodes inside networks is expected to continue in next generation networks.

Diversified network services in data path are a stark contrast to the current Internet architecture, which has been designed to specifically simplify the data path of routers. Such simplicity has allowed the implementation of high performance packet forwarding engines with application specific integrated circuits (ASIC). However, with diverse services, all-in-one ASIC approach is not practical due to its long design cycle and lack of flexibility to upgrade. This leads to a strong demand in router designs with programmable packet processing engines that can adapt to new functional requirements.

Typical programmable router systems use embedded multi-core systems-on-a-chip (MPSoC) in the form of network processors (NP) to provide a programmable environment for high-performance packet handling. Network applications execute in parallel across processors to exploit inherent parallelism present in packets from various connections. In the mean time, for highly isolated functionalities that are not suitable to be implemented in software (e.g. encryption and decryption in IPsec), ASIC in the form of hardware accelerator is widely adopted to increase overall functionality performance.

Although NP and hardware accelerator have received great research attention and their raw performance has been increasing dramatically, there are still challenges in programmable router systems revolving around adaptability to changing traffic workloads:

- Hardware Resources Utilization: One of the main challenges in software router design is the inherent tension between raw system performance and relatively low utilization of hardware resources. With the advent of MPSoC, the utilization has become a dominant factor that affects overall system performance.
- Dynamic Adaptation to Network Traffic: Dynamic adaptation is crucial for packet processing systems. The processing workload required by network traffic cannot be known in advance since end-systems may send packets to any arbitrary destination using any protocol. Thus, a packet processing system needs to either (1) overprovision for any possible traffic scenario or (2) dynamically adapt. With an increasing diversity of services that are provided in packet processing systems, the first choice is becoming less feasible.

In order to address these challenges, we introduce a novel design of a runtime environment that dynamically obtains profiling information from software applications and distributes workload across processing resources according to hardware capacities. We also present a programming interface that facilitates the development of applications on software routers. In particular, the contributions of this paper are as follows:

- Runtime Environment for Programmable Packet Processing Systems: Runtime environment acts as hardware resource manager in packet processing systems. By periodically obtaining profiling information from running applications, runtime environment is capable of distributing processing workload across available computation resources. This is critical for software applications to maximize underlying hardware utilization, which in turn leads to maximum performance of software routers.
- Programming Interface for Packet Processing Applications: A programming interface presents a unique hardware abstraction on different platforms. It hides the complicated details of underlying hardware, therefore simplifies development of applications as well as increases software portability.

The remainder of the paper is organized as follows: Section II presents related work. Section III discusses programming packet processing architectures and the software application model used for workload analysis. Section IV introduces our runtime environment as well as the programming interface

for software router applications. Section V summarizes and concludes the paper.

II. Related Work

Extensions to the feature set of the original Internet architecture have been proposed in the form of specific solutions (e.g., firewalls [1] or NAT systems [2]) or general concepts for extensibility. Networking paradigms that support extensibility have ranged from completely dynamic and user-programmable active networks [3] to a more controllable choice of predefined features in programmable routers [4]. Programmability of networks as service abstraction has been studied in active networks [5], [6] and on general platform [7]. On end-systems, support for extensibility is provided though configurable protocol stacks [8], [9]. In the context of next-generation Internet architectures, network services are used to provide flexible data path processing [10], [11]. To manage the complexity of new networking features, an IETF working group has attempted to define Open Pluggable Edge Services (OPES) [12].

We envision that network services can be implemented on a variety of platforms ranging from workstations routers [13] to programmable routers [14] and virtualized router platforms [15]. For next-generation networks, it is important to consider network services that range from application-layer end-system service (e.g., services in a grid computing environment [16]) to packet processing services on routers (e.g., data path services for the next-generation Internet [10]) to ensure flexibility to adapt to novel network uses that we cannot yet predict.

The use of specialized hardware in conjunction with general-purpose processors is common in the networking domain (e.g., lookups with TCAMs [17], FPGAs [18], [19]) as well as in other domains (e.g., media processing [20]). In general, hardware accelerators are desirable since they can perform functions faster, using less chip area and consuming less power than an equivalent software implementation on a general-purpose processor. On network processors, hardware support for specialized functions has been studied [21] and deployed [22]. The question of heterogeneity in processing systems has been studied in the general-purpose processor domain, where hardware designs are optimized to meet software requirements [23].

III. Network Services and Packet Processing Systems

Network service is a collection of functional components that are able to receive, store, process, and transmit data streams in the network. Besides traditional network service of IP forwarding, packet processing systems are required to accommodate more services. While switching fabrics provide fundamental connectivity between input and output ports, it is the packet processing systems that perform all protocol processing operations (typically on the input port). When considering router design for next-generation networks, much of the fundamental functionality of packet I/O and the switching fabric can remain unchanged. The packet processing system, however, is most affected by changes in the network architecture since packet processing requirements change. Thus, we focus on this aspect of router design.

A. Trends in Next Generation Networks

We observe the following trends that directly impact packet processing systems. We can expect that these trends continue in the foreseeable future.

- Network services increase in diversity and number. With network connectivity being deployed to an increasing number of diverse end-systems (e.g., cell phones, sensor networks, mobile ad-hoc networks, etc.), the types of communication paradigms that need to be supported will increase.
- Network virtualization increases feature sets of routers. The diversity in network services will not lead to the design of many different router systems. Instead, a large number of features will be deployed in different slices of one (or a few different) virtualized router systems. Thus, the overall feature set that needs to be supported on a router is expected to grow.
- Embedded packet processing system will become highly parallel. Advances in semiconductor manufacturing will increase the number of parallel processor cores on an MPSoC to several dozens and even hundreds within a few years.
- Network virtualization adds to dynamics in network. Changes in network traffic cause changes in the workload of packet processing systems. These dynamics will further increase as virtualization may cause more frequent changes of the types of processing features required on a router.

These observations lead to the question of what packet processor system designs should look like. In this section, we first illustrate the modeling of network services and reveal the behavior of applications of next generation networks. Then we briefly discuss what design options about system architecture are feasible.

B. Modeling Packet Processing Services

The key question that needs to be answered is how frequently different protocol processing features are used. If there are some features that are used very frequently, then an implementation with specialized hardware is justified. (Note that the computational complexity of the feature also plays a role in this decision, but we ignore this issue for now to simplify the discussion.) To answer this question, we use measurement results from a current router system and attempt to extrapolate to next-generation networks.

We have implemented a typical router configuration with the Click modular router. In Click (as well as in other router implementations), several protocol features are used to implement a complete protocol service (e.g., IP forwarding consists of several features including address lookup, TTL and header checksum adjustment, etc.). We have extended the system

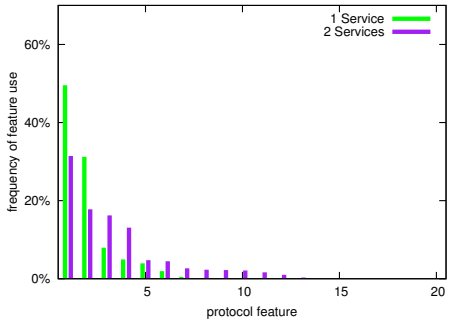


Fig. 1. Measured Frequency of Protocol Feature Use on Current Internet Router.

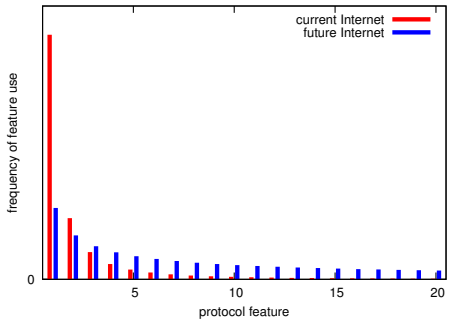


Fig. 2. Extrapolated Frequency of Protocol Feature Use on Next-Generation Network Router.

to provide profiling information on how frequently a Click element (i.e., a protocol feature) is invoked. The data obtained through measurement are shown in Figure 1. The Figure shows two scenarios: In the first case (1 service), the router performs only IP forwarding. In the second case (2 services), the router performs forwarding and IPSec processing.

It can be observed that some protocol features are used much more frequently than others. As indicated in the figure, the frequency of use of a protocol determines if it is implemented in the fast path or in the slow path. Another observation is that more services (i.e., more diversity in the data path) lead to more features being utilized. A side effect is a drop in the peak usage frequency of the more commonly used features. These observations lead to the following hypothesis:
Hypothesis. *Protocol processing features in routers are used with frequencies that follow a Zipf distribution.*

A Zipf distribution [24] is a representative of discrete power law probability distributions that are commonly encountered in networks and other domains. At this point, we only have the data shown in Figure 1 as empirical evidence that the hypothesis may be correct.

Assuming the above hypothesis' correctness, we can extrapolate the feature distribution to future network architectures. As we have argued above, next-generation networks are characterized by a diversity of protocols and services. Thus, we expect the trend that we have observed from 1 service to 2 services in Figure 1 to continue. Using a Zipf distribution as a basis, we thus get the feature use distribution shown in Figure 2.

C. Packet Processing System Architecture

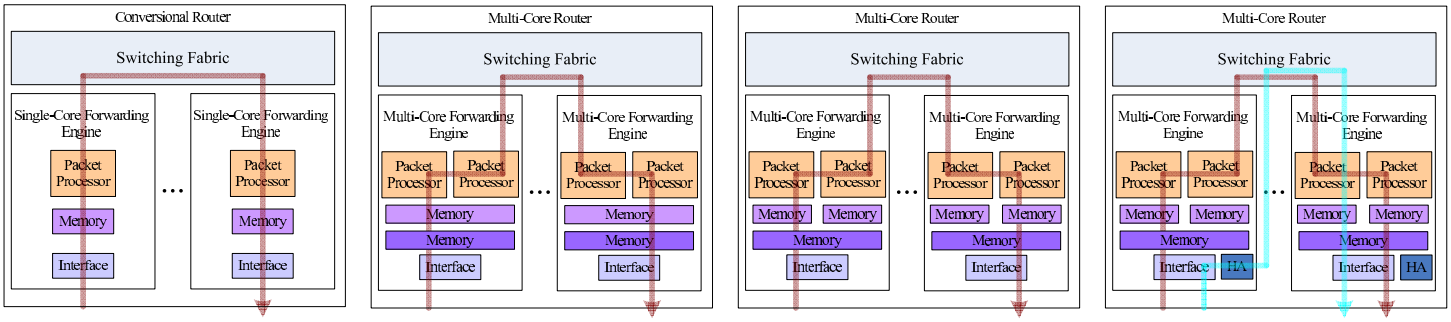
Flexibility is the dominant reason behind programmable packet processing systems. But in order to be applicable in practice, the performance of software routers should not be left too far away behind that of hardware routers. Besides the ever increasing I/O connection speed, there are three emerging trends that appear on software router platforms:

- Multi-Core Processor: Multi-core processors are adopted to execute software network applications in parallel, which in turn exploit the inherent parallelism present in packets from various connections. This trend is demonstrated by the continuously increasing number of cores that are embedded in single chip.
- Hybrid of Distributed and Shared Memories: Competition for shared memory bandwidth has a negative impact on multi-core system performance. Distributed memory is therefore adopted to provide separate storage for individual processing cores.
- Hardware Accelerator: Highly dedicated functionalities (e.g. Encryption operations) which are much less efficient in software are often implemented in the form of ASIC. The type of design is normally developed in the form of isolated IP core, therefore is separated from overall architecture and can be reused in different platforms.

These hardware components greatly increase raw performance of packet processing system. However, runtime system performance is not only determined by raw performance, but also affected by utilization of each component. We show several routers designs with differing architectures in Figure 3, and focus on the utilization problem individually.

1) Conventional Single Core Software Router: The architecture of a traditional software router that implements IPv4 is shown in Figure 3(a). As illustrated, packets are processed in a single core processor that executes specified IPv4 forwarding programs. Shared memory in this router system serves for both program data and packet storage. On such a system, performance will be maximized if all the bandwidths of processing, memory and network I/O match each other. Otherwise, the slowest component becomes the performance bottleneck.

2) Homogeneous Multi-Core Router with Shared Memories: Multi-core router with shared memories is shown in Figure 3(b). This router contains multiple levels of shared memories, which is a widely accepted implementation as it combines fast small sized memories for processing and slow large memories for packet storage. With multiple choices on both processor and memory space, utilization of underlying hardware is much more complicated than that of conventional router. On such systems, utilization problem of processing and storage hardware can be studied separately. Multi-core processor utilization could be mapped to several well defined algorithm problems. We have developed algorithm for workload distribution in our recent work [25] [26]. Memory subsystem utilization can be modeled and solved as knapsack problem.



(a) Router with Single Processor and Shared Memory.

(b) Router with Multi-core Processors and shared Memories.

(c) Router with Multi-core Processors and Hybrid Memories.

(d) Router with Multi-core Processors, Hybrid Memories and Hardware Accelerator.

Fig. 3. Architecture of Routers with Different Packet Processing Systems.

3) Homogeneous Multi-Core Router with Hybrid Memories: As shared memory leads to competition among processing cores, distributed memories are introduced to meet the need for memory bandwidth in multi-core systems with more and more embedded cores. Figure 3(c) shows the high level architecture of this type of routers. Unlike shared memory, utilization of hybrid memories is closely related to distribution of processing tasks on multiply cores. Therefore it is an unsolved problem that still under research.

4) Heterogeneous Multi-Core Router with Hybrid Memories: Figure 3(d) shows a router with multi-core processor, hybrid memories and hardware accelerators. As mentioned above, specific functionalities such as computation intensive modules are often implemented as ASIC to achieve better performance than plain software approach. On such systems, hardware accelerator should be utilized whenever possible for certain benefits (e.g. power consumption and performance).

With resources increasing in terms of both numbers and types, the utilization of resources becomes a fundamental problem that must be addressed to make software router practical. Various services that need to be deployed on routers further deteriorate this problem since multiple instances of different applications that run in parallel make overall utilization more difficult to control. Therefore, it is necessary to have a layer of system software to manage overall resource utilization.

IV. Programming Abstraction and Runtime Environment

As discussed above, dynamic adaptation is critical to packet processing systems. In this section, we first discuss the programming abstractions which serve the purpose of hiding underlying hardware details from packet processing applications. Then we introduce the runtime environment that dynamically distributes application workload and storage across available resources. Figure 4 shows the architecture of our system. As illustrated in Figure 4, the system can be divided into two parts: offline application development and runtime operation. Programming abstractions help in the offline development design. Runtime control and profiling information collection is done in the online portion of the system.

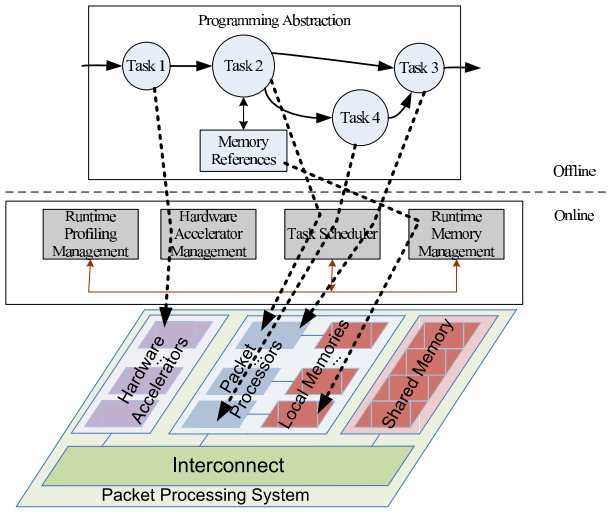


Fig. 4. Runtime Environment.

A. Programming Abstractions

Programming abstraction serves as the base for packet processing application development. At highest level, it provides two aspects that facilitate software development: application model and programming interface. Figure 5 illustrates an IPv4 forwarding service built with programming abstraction.

The application model defines the structure of application. In our model, applications are divided into processing steps (round nodes in Figure 5) and data set (square nodes in Figure 5). A processing step consists of a set of instructions that perform certain functionality, and is therefore called "task" in this paper. Pseudo C code for the task of "IP Lookup" in Figure 5 is shown as follows:

```
struct IP_Lookup{
    ...
    unsigned long service_time;
    unsigned long utilization_rate;
    boolean *process_function();
    ...
}
```

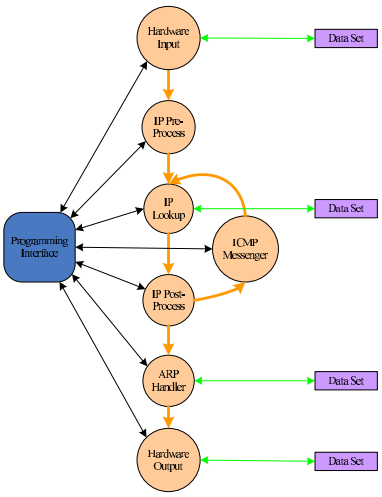


Fig. 5. Programming Abstraction for Packet Processing Applications.

```
boolean process_function(){
    ...
    packet=get_packet();//library call
    ...
    look_up(packet);     //data set access
    ...
    val=send_packet();  //library call
    return val;
}
```

Task represents the processing requirement of an application, and data set represents the storage requirement. A pseudo C code for a typical data set is shown below:

```
struct data_set{
    ...
    unsigned long capacity;
    unsigned long access_rate;
    unsigned char * buffer; //pointer to
                            //storage
    ...
}
```

Tasks must be carried on processing units, and their execution contributes to workload on processing units. Data set need to be allocated to memory subsystem, and the reference to data sets consumes memory bandwidth. Communications between tasks (lines between tasks in Figure 5) represent the flow of network packets. Such communication varies from system to system (e.g. shared memory, next neighbor register), therefore is implemented in programming interface (ellipse node in Figure 5).

Programming interface provides a unique set of interface to various applications on different hardware platforms. It plays a similar role as system call in a general Linux system. We observe that three types of interfaces are necessary in packet processing programming interface:

- Memory Management: Memory should be used in a controllable way for system to monitor, profile and utilize memory resources. As software router systems are equipped with at least one form of memory, the ability to hind memory interface details from application is critical to programming interface.
- Inter-Task Communication: In general, inter communications between tasks can be implemented in several ways. For example, shared memory on general x86 systems and next neighbor register on IXP 2400 can both be used as communication channel.
- Hardware Device Management: Except for above mentioned hardware accelerators, there are common hardware devices on packet processing systems, such as network interfaces, hardware timers, etc.

B. Runtime Profiling Management

Runtime profiling module collects profiling information about tasks and data sets at runtime. It monitors the dynamic trends in the processing workload and memory access. Profiling parameters, which are critical for function of runtime environment, are embedded in the definition of task and data set. In this paper, we define "service time" and "utilization rate" for tasks, "capacity" and "access rate" for data sets. All profiling information is updated periodically and stored in runtime profiling manager, to be used by other management components in runtime environment.

An important question is how frequently to update this information and how frequently to revise the distribution. The utilization information should be collected over a reasonably large number of packets to average out short packet bursts that are not representative of the overall workload. The interval between task mappings should depend on how much the workload changes. In related work, different mechanisms have been proposed in this context (e.g., mapping based on length of inter-processor queues [27], mapping based on fixed intervals optimized for workload [28]).

C. Task Scheduler

Task scheduler uses the dynamic profiling information as the guide to distribute tasks across processing units. For example, in a homogeneous multi-core system, the distribution process is often modeled as Bin Packing problem. Therefore the objective is to achieve balanced workloads across all processing cores. This step is critical for computation intensive application such as IPsec. We have studied task workload in [25] and implemented dynamic task distribution with Click model in [26].

D. Memory Management

Memory management component works on data sets. Packets and applications' data both need to consume memory storage space. In multi-level hybrid memory subsystem, the utilization of bandwidth and capacity will determine execution speed of applications, especially for memory access intensive services such as IP forwarding and Firewall. Multiple shared

memory layers could be modeled as a 0-1 Knapsack problem, which addresses the question of how to maximize bandwidth utilization within capacity limits for each memory layer.

E. Hardware Accelerator Management

Hardware accelerator management component maps tasks that need services from specific accelerator to underlying hardware. As different accelerators vary greatly in the way of controlling and exchanging data, this component needs to be tuned for each real system.

Above all, implementation of such runtime environment executes in control processor of programming packet processing systems. As the administrative layer of software, runtime environment is able to monitor the activities of applications, therefore attain profiling information. However, in order for the runtime environment to work, it needs to take over the hardware resources management from application. Otherwise individual application may have the power to affect overall system performance. For example, memory leaks in one application may use up all memory resources therefore leaving other applications without enough resources to execute. To address this problem and hide underlying hardware details from applications, a programming interface that remains unique across different programmable packet processing system is necessary.

V. Conclusion

In this paper, we have presented a discussion about programmable packet processing systems. We have introduced programming abstractions which can not only help the development of network services but also allow quantitative analysis on services workload. Based on these abstractions, we have devised a runtime environment which obtains profiling information at runtime and distribute computation dynamically across available hardware resources. This work presents a design that provides a first step towards a runtime system support for heterogeneous packet processing systems where processing and memory management are considered.

References

[1] J. C. Mogul, "Simple and flexible datagram access controls for UNIX-based gateways," in *USENIX Conference Proceedings*, Baltimore, MD, Jun. 1989, pp. 203–221.

[2] K. B. Egevang and P. Francis, "The IP network address translator (NAT)," Network Working Group, RFC 1631, May 1994.

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

[4] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Proc. of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.

[5] T. Fuhrmann, T. Harbaum, M. Schöller, and M. Zitterbart, "AMnet 2.0: An improved architecture for programmable networks," in *Proc. of the International Workshop on Active Networks (IWAN)*, Zurich, Switzerland, Dec. 2002.

[6] L. Ruf, R. Keller, and B. Plattner, "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, Jul. 2004.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[8] N. T. Bhatti and R. D. Schlichting, "A system for constructing configurable high-level protocols," in *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, Cambridge, MA, Aug. 1995, pp. 138–150.

[9] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture," *SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 17–22, Jan. 2003.

[10] T. Wolf, "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.

[11] S. Ganapathy and T. Wolf, "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.

[12] A. Barbir, P. Reinaldo, R. Chen, M. Hofmann, and O. Hilarie, "An architecture for open pluggable edge services (OPES)," Network Working Group, RFC 3835, Aug. 2004.

[13] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64–76, Jan. 1991.

[14] L. Ruf, K. Farkas, H. Hug, and B. Plattner, "Network services on service extensible routers," in *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)*, Sophia Antipolis, France, Nov. 2005.

[15] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[16] I. Foster and C. Kesselman, Eds., *The Grid – Blueprint for a New Computing Infrastructure*, 2nd ed. Morgan Kaufmann, 2004.

[17] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 560–571, May 2003.

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

[19] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *Proc. of Eighth International Symposium on Field-Programmable Gate Arrays*. Monterey, CA: ACM, Feb. 2000, pp. 137–144.

[20] M. J. Rutten, J. T. J. van Eijndhoven, E. G. T. Jaspers, P. van der Wolf, E.-J. D. Pol, O. P. Gangwal, and A. Timmer, "A heterogeneous multiprocessor architecture for flexible media processing," *IEEE Design and Test*, vol. 19, no. 4, pp. 39–50, Jul. 2002.

[21] M. Grünewald, J.-C. Niemann, M. Porrmann, and U. Rückert, "A mapping strategy for resource-efficient network processing on multiprocessor socs," in *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, Paris, France, Feb. 2004, pp. 758–763.

[22] *Intel IXP2800 Network Processor*, Intel Corporation, 2002, http://developer.intel.com/design/network/products/npfamily/ixp2800.htm.

[23] J. M. Paul, D. E. Thomas, and A. Bobrek, "Benchmark-based design strategies for single chip heterogeneous multiprocessors," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, Stockholm, Sweden, Sep. 2004, pp. 54–59.

[24] G. K. Zipf, *The Psycho-Biology of Language*. Boston, MA: Houghton Mifflin, 1935.

[25] Q. Wu and T. Wolf, "Dynamic workload profiling and task allocation in packet processing systems," in *Proc. of IEEE Workshop on High Performance Switching and Routing (HPSR)*, Shanghai, China, May 2008.

[26] ——, "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.

[27] R. Kokku, T. Riché, A. Kunze, J. Mudigonda, J. Jason, and H. Vin, "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.

[28] T. Wolf, N. Weng, and C.-H. Tai, "Run-time support for multi-core packet processing systems," *IEEE Network*, vol. 21, no. 4, pp. 29–37, Jul. 2007.