Challenges and Applications for Network-Processor-Based Programmable Routers

Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
wolf@ecs.umass.edu

***Abstract*— The growth in ubiquity, performance, and commercial and academic use of computer networks continues to demand more performance and flexibility from the underlying network infrastructure. The lack of security, quality of service, and manageability in the current Internet poses a significant challenge and highlights the importance of considering other network designs. Programmable routers provide a vehicle for experimenting with new architectures that allow the dynamic deployment of new protocols and services. To achieve the necessary throughput performance, programmable routers employ network processors, which are embedded system-on-a-chip multiprocessors. This paper discusses the challenges that such systems pose in terms of system architecture, programming abstraction, and deployment. The potential applications highlight the benefits of making router programmability a first-class networking function.**

I. Introduction

Several shortcomings of the design of today's Internet are expressing themselves in the lack of flexibility, security, and manageability. When the Internet was originally designed in the 1970's, the goal was to have a simple, packet-switched communication infrastructure, which connects a large number of systems through routers [1]. The network itself was kept relatively simple and provided basic communication between the end-systems. This led to networking protocols, where most of the complexity is implemented on the end-systems (e.g., retransmission of lost packets, congestion control based on round-trip time measurements).

Over time, several additions have been proposed and implemented in the Internet, because the initial design did not consider them. Since the Internet does not support the dynamic deployment of new protocols, these features needed to be added as special processing functions to routers. The following list highlights a few, which are characterized by the need of support by the network (i.e., they cannot be implemented on end-systems only):

- Random Early Detection (RED [2]) is a queue management scheme for routers to fairly drop packets from rogue TCP flows. This is can be implemented in a very simple fashion, but it constitutes a type of processing on a router. Almost all current routers implement some form of RED, but only few use it in practice.
- Network address translators (NAT [3]) are another common component in IP networks. A NAT allows multiple hosts in a stub domain to use a single globally unique IP address. IP packets passing between the stub domain and the Internet are modified by the NAT. This reduces the number of IP addresses used by a stub domain and thereby extends the time before all IP addresses are assigned.
- Firewalls [4] are a standard security component of most networks. Packets are filtered depending on rules defined by the network administrator. This enables the blocking of network traffic that could compromise the security of hosts on the network (e.g., port scanning). The firewall rules can be numerous and complex, which requires significant computational power on the firewall to keep up with typical access link speeds.
- Intrusion detection systems (e.g., snort [5]) check packet headers and content against signatures of well-known attacks. Traffic the is considered malicious can be blocked dynamically.
- Web switching [6] is a method of distributing a web server over several physical machines while presenting a single font-end to the outside. Web switches parse HTTP requests in packets and determine the appropriate server to which to forward the request. Since the HTTP request is sent only after the TCP connection is established, the web switch also has to splice the TCP connection between client and back-end server.

In practice, these changes either have been slowly added to all routers (e.g., RED) or they were implemented on servers that are connected to the routers (e.g., intrusion detection). These specialized solutions are working reasonably well in practice, but also cause the Internet architecture to diverge more and more from its original design as more and services require processing inside the network (see Figure 1).

Over the coming years, it can be expected that the Internet design will need to be adapted further. The expansion of network connectivity to mobile and ad-hoc sensor networks with end-systems with very low processing power (e.g., RFID tags) will bring about considerable changes. In order to test and deploy new ideas in the Internet, it is necessary to develop suitable supporting infrastructure. A key component of this infrastructure is the programmable router. Introducing programmability into the data path of routers provides flexibility to adapt to new protocols and services is by means of simple changes in software.

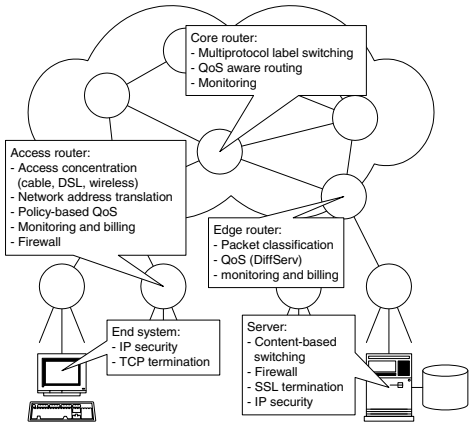


Fig. 1. Network Services Requiring Processing Inside the Network.

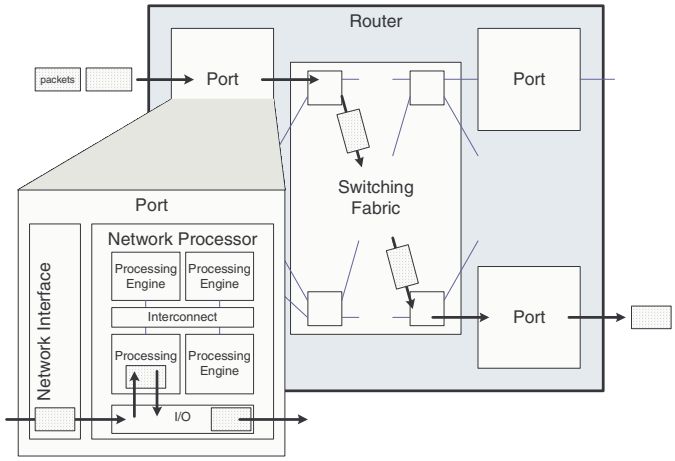


Fig. 2. Programmable Router System.

II. Programmable Routers

A key infrastructure component for a more flexible and dynamic network are programmable routers, which are characterized by having general-purpose processing capabilities in their data path. These processing units can be programmed to perform various protocol operations as well as complex payload processing. Unlike with traditional routers, deployment of new protocols can be achieved by reprogramming the system rather than exchanging expensive hardware. This programmability of the data plane extends the traditional store-and-forward paradigm of routers to store-process-and-forward. The processing step is where interesting new services and protocols can be implemented in the network. It also provides the means for a unified architectural view of additional network features that are already implemented (see Section I).

A. Software Issues

Making network systems programmable raises many security and manageability questions. Often, software programmability of the data path is equaled with "active networking" [7], [8], which allows the programming of processing steps by the end-system application. It is important to note that there are a variety of other mechanisms for controlling programmability that do not require such open assumptions. A more likely scenario for initial deployment is that programmable routers can be updated by system administrators or router vendors to install well-tested new services. An example for such an approach are open pluggable edge services (OPES) [9]. Such a model might later migrate to a more open programming platform as proposed by the active networking community, but this is not a necessary requirement.

B. Hardware Issues

The flexibility of a programmable router comes of course at a price. Software processing is inherently slower than customized logic that is optimized for protocol processing. To address this problem, high-performance embedded processing systems, called "network processors" (NPs) have been developed. Network processors have been implemented as highly parallel multiprocessor systems-on-a-chip that are optimized for high-bandwidth I/O tasks (see Figure 2).

The key idea to achieving the necessary processing performance is to exploit the parallelism that is inherent to network systems and far exceeds that of traditional workstation processor systems. There are three layers at which parallelism can be exploited in a network multiprocessor:

- Flow Level. Packets from different flows do typically not interact with each other. Therefore they can be processed completely independently.
- Packet Level. In many protocols, there is no dependency among packets within a flow. For example, simple IP protocol forwarding does not keep or modify state between packets. Therefore packets can be processed in parallel, even if they belong to the same flow. In some cases it is necessary to ensure that the original packet order is restored after processing.
- Instruction Level. When processing a packet, there are several ways that parallelism can be exploited. These approaches are the same as found in traditional processor architecture. In particular, pipelining and instruction-level parallelism can be used.

Parallelism in the processing workload can be translated to parallelism in a processing system. This enables the design of processing engines with a large number of independent processors with less need for tight interactions and synchronization between them.

Commercial examples of network processors are the Intel IXP2400 [10], the AMCC np7510 [11], the EZchip NP-1 [12], and the Agere Fast Pattern Processor and Routing Switch Processor [13]. The number of embedded RISC processor cores ranges from as little as eight in the IXP2400 to an amazing 188 in the Cisco Silicon Packet Processor (SPP). Designing high-performance network processors as well as programming and managing them during run-time raises a number of interesting scientific and engineering challenges.

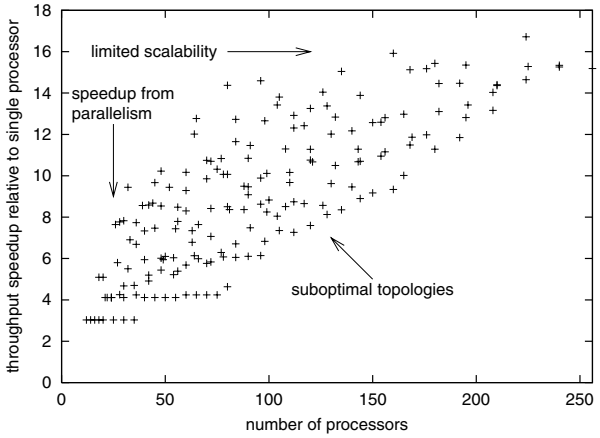


Fig. 3. Topology Impact on Performance.

III. NP System Challenges

A. NP Design

The system design of an embedded multi-processor network processor has a large number of degrees of freedom. In practice, commercial network processor designs have not converged on roughly the same system architecture the way workstation and server processors have. There are fundamental differences in terms of system topology, memory design, and the way coprocessors for hardware acceleration are used. The first-level design choices that need to be considered are:

- System topology: How should processors be interconnected? Should they be arranged in a fully parallel or pipelined configuration?
- Memory configuration: How many interfaces for each type of memory (SRAM and DRAM) should be used? Should all processors have access to all memory?
- Special features: What functions should be accelerated in hardware (e.g., route lookups, checksum computation)? Should there be programmable logic for hardware acceleration? What are special instruction set extensions that are beneficial for network processing?

These choices have significant impact on the performance of the network processing system. To illustrate this point, we have used an analytic performance model to determine the performance of different NP systems [14]. Figure 3 shows the performance of different system configurations as a function of the number of processors used. It clearly shows that some configurations perform much better with the same number of processors than others. Another important observation is that speedup is constraint by limited scalability. This is caused by contention on the memory interfaces. Overcoming the bottlenecks of memory systems in network processors has received much attention recently and different design alternatives have been explored [15].

B. Programming

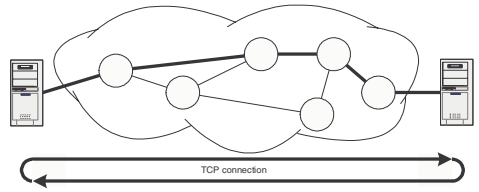
With current software development tools and methodologies, it is challenging to program an application for an NP. Most NP vendors provide software development kits for their architecture that use modular programming abstractions. A major focus is on software reuse across different NP platforms [16]. However, the multiprocessor nature of an NP requires that the application developer partition the application and allocate it to processing resources. To achieve better load balancing for higher throughput, this typically requires that parts of the application are programmed in assembly and fine-tuned for performance. The major drawback of most of these approaches is that they require the application developer to have an in-depth understanding of the NP system architecture. This will become an increasingly more pressing problem as network processors—as well as other embedded systems—move towards parallel architectures with dozens of parallel processing resources. In such systems, the allocation of processing tasks to resources needs to be done in an automated fashion without requiring the developer to make such decisions.

We address this problem with a "bottom-up" approach that differs from existing work insofar that we use profiling information from a simple uniprocessor implementation of an NP application. This profiling information is used to extract all available parallelism in the application, to map the application to processing resources, and to model the run-time performance through an analytic performance model [17]. This bottom-up approach of analyzing and mapping workloads promises to significantly reduce the complexity the application developer has to deal with.

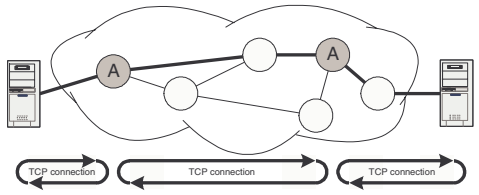
C. Run-Time Management

Many current network processor systems only support one static workload scenario at a time. Changes to the workload during run-time causes the performance to degrade as processing deviates from the fine-tuned implementation (e.g., imbalance in the software pipeline). Thus, it is very difficult to integrate and dynamically change multiple packet processing functions on a single network processor. However, network processing is inherently a dynamic process [18]. The main motivation for implementing packet processing functions in a network processor (rather than in a faster, more power-efficient custom logic device) is the need to change the functionality over time. Changing traffic patterns, new network services and protocols, new algorithms for flow classification, and changing defenses against denial of service attacks present the dynamic background that a programmable router needs to accommodate. This requires that the router (1) can implement multiple packet processing applications at the same time, (2) can quickly add and remove processing functions from its workload, and (3) can ensure efficient operation under all circumstances. In particular, the management of various system resources is important to avoid performance degradation from resource bottlenecks.

We have explored the design considerations for network processor operating system in this context [19]. The computational complexity of finding an optimal software configuration for a given workload is considerable. Therefore, there is an



(a) Conventional TCP Connection



(b) Accelerated TCP Connection

Fig. 4. Conventional and Accelerated TCP Connections using Programmable Routers.

inherent tradeoff between optimizing one particular software setup and the ability to adapt quickly to changes in network traffic.

IV. Applications

The ability to change the data path processing in a router allows the implementation of a variety of novel network functions. These "applications" of network processing aim at providing novel functionality as well as improved performance. They also provide a unified platform for already existing network extensions discussed in Section I.

Another example for a novel application that targets performance improvements in the network through the use of network processing is "Transparent TCP Acceleration" [20]. Programmable routers in the network can opportunistically act as TCP proxies by terminating TCP connections and opening a new connection towards the destination. This is done transparently (i.e., undetectable to the end-system) and effectively converts one connection with a large round-trip time into multiple connections with smaller round trip times. Since the throughput of a TCP connection is inversely proportional to the round-trip time, accelerated TCP connections achieve higher throughput. Under congested conditions, accelerated TCP increases the utilization of the bottleneck link since it can retransmit lost packets locally and avoids oscillations, which are often caused by multiple connections with long round-trip times backing of in sync.

Looking forward, programmability on routers can provide the means for network service providers to differentiate their offerings from those of the competition. Programmable routers can provide premium services that enhance the functionality and performance of data transfers. It is also conceivable that they can offer services that are traditionally limited to the end-system (e.g., virus scanning not only for downloaded data, but also for files stored on the end-system that could be streamed through the access network).

V. Conclusion

The field of programmable routers is still in its early stages. It can be expected that there will be much growth in this area as network functionality needs to be extended to address exiting and novel challenges. Programmability in the data path is a key enabling technology for future network architectures that can provide services that go beyond simple packet forwarding.

References

[1] D. D. Clark, "The design philosophy of the DARPA internet protocols," in *Proc. of ACM SIGCOMM 88*, Stanford, CA, Aug. 1988, pp. 106–114.

[2] S. Floyd and V. Jacobson, "Random early detection (RED) gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

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

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

[5] *The Open Source Network Intrusion Detection System*, Snort, 2004, http://www.snort.org.

[6] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, "Design, implementation and performance of a content-based switch," in *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 1117–1126.

[7] D. Wetherall, U. Legedza, and J. Guttag, "Introducing new internet services: Why and how," *IEEE Network*, vol. 12, no. 3, pp. 12–19, May 1998.

[8] 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.

[9] *Open Pluggable Edge Services*, IETF, 2003, http://www.ietf-opes.org/.

[10] *Intel Second Generation Network Processor*, Intel Corporation, 2002, http://www.intel.com/design/network/products/npfamily/ixp2400.htm.

[11] *np7510 10 Gbps Network Processor*, AMCC, 2003, http://www.amcc.com.

[12] *NP-1 10-Gigabit 7-Layer Network Processor*, EZchip Technologies Ltd., Yokneam, Israel, 2002, http://www.ezchip.com/html/pr_np-1.html.

[13] *PayloadPlus^{TM} Fast Pattern Processor*, Lucent Technologies Inc., Apr. 2000, http://www.agere.com/support/non-nda/docs/FPPProductBrief.pdf.

[14] N. Weng and T. Wolf, "Pipelining vs. multiprocessors - choosing the right network processor system topology," in *Proc. of Advanced Networking and Communications Hardware Workshop (ANCHOR 2004) in conjunction with The 31st Annual International Symposium on Computer Architecture (ISCA 2004)*, Munich, Germany, June 2004.

[15] J. Mudigonda, H. M. Vin, and R. Yavatkar, "Overcoming the memory wall in packet processing: Hammers or ladders?" in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, Princeton, NJ, Oct. 2005.

[16] S. D. Goglin, D. Hooper, A. Kumar, and R. Yavatkar, "Advanced software framework, tools, and languages for the IXP family," *Intel Technology Journal*, vol. 7, no. 4, pp. 64–76, Nov. 2003.

[17] N. Weng and T. Wolf, "Profiling and mapping of parallel workloads on network processors," in *Proc. of The 20th Annual ACM Symposium on Applied Computing (SAC)*, Santa Fe, NM, Mar. 2005, pp. 890–896.

[18] 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.

[19] T. Wolf, N. Weng, and C.-H. Tai, "Design considerations for network processor operating systems," in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, Princeton, NJ, Oct. 2005, pp. 71–80.

[20] T. Wolf, S. You, and R. Ramaswamy, "Transparent TCP acceleration through network processing," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, St. Louis, MO, Nov. 2005.