Convergence of Communication and Processing in Next-Generation Networks

Tilman Wolf and Soumya Mahadevan
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA
{wolf,somahade}@ecs.umass.edu

***Abstract*—We argue that applications can benefit from the offloading of processing tasks into the data path of next-generation networks, where custom packet processing functions can be implemented. We present the concept of application-layer services and how they can be integrated into our existing network service architecture for easy use and control. Using a web server application as an example, we discuss how an application can be partitioning between end-system and offloading tasks. Our prototype implementation based on an Intel IXP2855 network processor demonstrates that web page access times can be improved by using our application-layer services.**

***Index Terms*—network service, network processor, offloading.**

I. Introduction

The networking community is in the process of developing an understanding of what the next-generation Internet architecture should look like. One of the key aspects of this effort is the clean-slate assumption that avoids the need for backward compatibility [1]. Many of the proposed architectures agree in their use of hardware virtualization [2]. An implicit requirement for dynamically creating virtualized network slices with as of yet unknown functionality is that they can be reprogrammed at runtime. Therefore, programmability in the data path of routers is widely considered an essential feature in next-generation network architectures.

When providing general-purpose programmability in the data path, a key challenge is to determine how to manage and control this functionality in a way that does not make the network too complex. The deployment of active networks, which provide the most general way of data path processing [3], was hampered by these difficulties. To balance the ability to deploy new packet processing features with manageable control, we have developed the concept of network services in prior work [4], [5]. Network services implement a set of well-known protocol processing features and can be combined in arbitrary sequence to set up custom connections. However, this service abstraction (as well as other system that permit runtime composition of protocols), focus mostly on network and transport layer functions.

In this paper, we argue that applications can benefit from offloading application-layer tasks into the network. This kind of application offloading can be supported by our network service architecture through the use of application-layer service. The resulting architecture presents a convergence of communication and processing in a single infrastructure. Our specific contributions are:

- Design of application-layer service abstraction that implements data path processing functions and provides balance between generality and usability.
- Discussion of application-layer offloading in the context of a web server example.
- Evaluation of a prototype system that uses an Intel IXP2855 network processor to offload web server processing.

The remainder of this paper is organized as follows. Section II discusses related work. A general architecture for data path processing in next-generation networks is presented in Section III. A specific example of application layer offloading in a web server application is discussed and evaluated in Section IV. Our work is summarized in Section V

II. Related Work

The original Internet architecture [6] does not permit packet processing features beyond those specified in the IP protocol [7]. To accommodate the need for support of new protocols and communication paradigms, programmability in the data path has been proposed. More recently, network virtualization [2], which is based on such programmability, has been developed in the context of next-generation network infrastructure. Programmability in the data path of networks has also been proposed in the form of active networks [3], where packets carry custom program code. Numerous active network platforms have been developed (e.g., [8]–[10]). The difficulties in managing the complexities of active networks has lead to programmable router designs [11], [12], where packets can choose from a set of predefined functions.

Abstractions for accessing programmability and custom composition of protocol functions in the network have been proposed in the form of configurable protocol stacks [13] and heaps [14]. Per-flow custom protocol stacks have been proposed in the context of next-generation networks [15]. Our previous work uses network services as abstraction [4], [5] and is discussed in more detail in Section III.

Offloading of processing tasks onto networking devices has been proposed for protocol processing tasks. Transport-layer TCP offloading has been commercially developed [16]. TCP onloading is used in high-performance end-system protocol processing [17]. Our work offloads entire application tasks and thus does not create the separation between protocol

processing and application that is caused by conventional offloading.

III. Processing in Data Path

Processing operations in the data path of networks are an important aspect of next-generation networks. We argue that such processing is not limited to just network-related tasks, but can also be used for application-layer tasks. The resulting architecture presents a convergence of communication and computation in the networking infrastructure.

A. Need for Data Path Processing

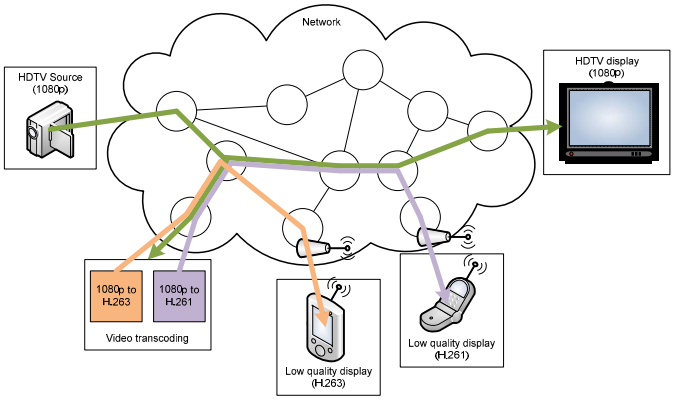
To motivate why processing in the data path of networks is essential, we provide two examples. One focuses on processing in the network and transport layer and the other one focuses on processing in the application layer.

1) Network and Transport Layer Processing: As the Internet becomes more ubiquitously deployed in the environment, more diversity in end-systems and communication paradigms need to be accommodated. For example, sensors and RFID tags require very different communication protocols than conventional clients and servers. However, all these systems need to be integrated in the same infrastructure (rather than building independent, specialized networks). Thus, next-generation networks have two fundamental requirements:

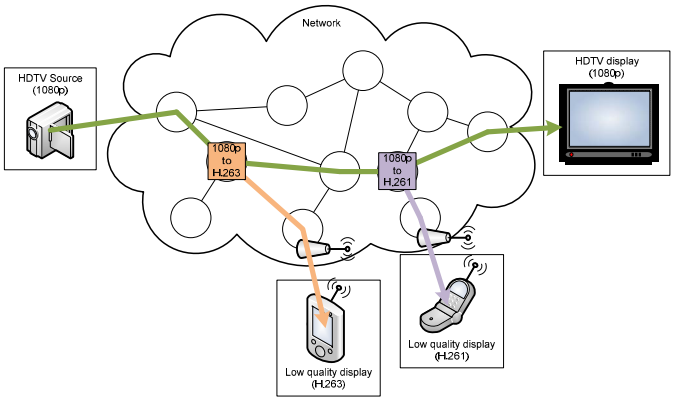
- Programmability: The functionality of the network cannot be limited to a single, fixed protocol stack. As new systems and protocols are added to the Internet, the infrastructure needs to be able to accommodate them. In particular, fixed-function ASIC forwarding engines need to be replaced by routers that can be reprogrammed after deployment. Programmability in router systems can be achieved by using multi-core embedded systems-on-chip (e.g., network processors [12]) or programmable logic devices (e.g., FPGAs [18]).
- Customization of data path at runtime: The processing performed in the data path of a router determines what protocols are being implemented in the system. To support utilizing programmability in router systems, it is important to provide a mechanism for setting up custom data-path operations at runtime. This dynamic customization can be done on a per-network level (e.g., using virtualization of network infrastructure and configuring different slices [2]) or on a per-connection level (e.g., custom protocol stacks [15]).

With this level of programmability and custom configuration, it is possible to deploy novel networking functionality. Examples are:

- Deployment of QoS scheduling in a network to support delay-critical connections.
- Deployment of new denial-of-service detection algorithms and countermeasures to squelch attack traffic close to the source.
- Deployment of opportunistic forward-error-correction to reduce packet losses on links with high bit errors.



(a) Video Distribution with Transcoding in Overlay Network.



(b) Video Distribution with Processing Services in Data Path.

Fig. 1. Example of Video Distribution on Different Network Architectures.

These examples operate on the network and transport layer of the network and thus impact network functionality for all applications that use the network (or all connections that are customized with these features).

2) Application Layer Processing: It is also possible to use processing functionality in the network that aims at the application layer. Thus, the processing is performed at the level of the packet payload (and thus includes transport and network layer. An example of why application-layer processing inside the network (rather than solely on end-systems) is useful is shown in Figure 1.

The example shows a scenario where video is distributed from a source on the left to three different types of devices. Each device requires a different video format due to their system characteristics (i.e., screen size, processing capabilities). Since the source sends the video in just one specific format (e.g., high-quality), it is necessary that the video stream be transcoded. That is, the application-layer data needs to be modified so that the low-end systems can display the video. This transcoding could be performed at three locations:

- Transcoding on the sender's server: It is difficult to envision that the server can transcode a video stream to match all possible end-system devices. Television broadcasts reach millions of users and the number of different end-systems could be very large. The server may not have all

the necessary codecs or the necessary bandwidth to send all possible video streams in parallel. Thus, this approach is not scalable.

- Transcoding on the receiver's client: The server could send the video stream in the highest available quality (to satisfy those users who can display high-quality video). Receivers that require lower quality could transcode this stream on the end-system to match the specific system characteristics. This approach ensures that a suitable codec is available. However, the bandwidth of the receiver may not be high enough to receive the high-quality stream (e.g., due to wireless links as shown in the figure). Also, the processing capabilities of the receiving end-system may not be high enough to perform the required transcoding. For receivers with low-quality displays, this solution wastes a lot of link bandwidth and processing resources.
- Transcoding in the network: The "sweet spot" for video transcoding is inside the network. By transcoding video before it is received by the client, it can be ensured that no bandwidth is wasted on the hops after transcoding and that the receiver's processing system is not overloaded. Also, if transcoding is performed by the Internet service provider, it is conceivable that it is known what end-systems are being used (e.g., a particular cell phone provider has a limited set of handheld devices that they sell to customers). Thus, the necessary codecs can be available.

The way in-network transcoding is currently implemented is shown in Figure 1(a). Since application-layer processing is limited to end-systems, video-transcoding must be on an end-system. For the reasons discussed previously, it cannot be done on the sender or receiver end-system. Thus, it is typically done on a third end-system. This approach creates an overlay network, with virtual links shown in different colors. One of the problems of this approach is that it also wastes some bandwidth and requires the sender to use the overlay network for transmission (rather than the conventional network).

To overcome these problems, data-path processing can be used as shown in Figure 1(b). In this case, the sender transmits traffic without being aware of any transcoding that may happen. The transcoding is performed in the network at the branching points of the multicast tree where downstream nodes of one branch require all the same video coding. The receivers receive the video stream in exactly the format that they need and do not need to perform any transcoding operation. Thus, this placement of the application-layer transcoding task is ideal. This example is only one of many that show that application-layer processing inside the network (i.e., in the data path or routers) is desirable.

B. Abstractions for Data Path Processing

To make data path processing accessible to the user (or to an end-system application), it is necessary to provide some abstractions that can be used to control the data path. In this context, there is a tension between two necessary requirements:

- Flexibility: Ideally, an end-system should be able to set up any arbitrary data path processing functionality to support the broadest possible range of functions.
- Manageability: Ideally, data path processing should be easy to use and generate little additional complexity in the system.

There are several approaches that differ in the way they balance these two goals:

- Traditional Internet: The Internet architecture kept most of the complex functionality on end-systems and tried to make the data path as simple as possible [6]. Thus, it has practically no flexibility, but is relatively easy to manage.
- Active networks: Active networks proposed to enclose in each packet a program that determines how the packet is processed on a router [3]. This approach provides maximum flexibility, but causes great problems in terms manageability. Router systems that can support this functionality are difficult to implement and the impact of executing arbitrary programs on routers is difficult to judge and control.
- Programmable networks: An intermediate solution is a network that uses programmable routers [12], where a set of custom functions is available on routers. End-system applications can chose which of these functions to use, but cannot deploy arbitrary, new functions themselves. Thus, some level of flexibility is provided, while also keeping the complexity of the system at bay.

Based on these observations, it can be seen that it is important not only to provide programmability in the data path, but also to consider how this functionality is exposed to end-system users and applications.

C. Network Service Architecture

In our prior work, we have addressed this problem of balancing the programmability in the data plane with ease-of-use in the control plane [4], [5]. The design of how to provide application-layer processing in the network is based on several concepts that we have developed in this work, and thus we review it here briefly. The main idea is to represent data-path processing functionality as a set of well-known "network services" that can be combined into a custom sequence for a connection. A network service encompasses the processing that is necessary for a complete protocol feature (e.g., network address translation, intrusion detection, QoS scheduling, reliability, privacy, etc.). Some services are stand-alone functions (e.g., anomaly detection); others come in pairs (e.g., encryption and decryption). When setting up a connection, the end-system application can specify a sequence of services that need to be performed between the sender and the receiver. The network determines a suitable route and placement of services and instantiates the connection accordingly.

The network service architecture that we have developed is shown in Figure 2 (from [5]). In the control plane, where

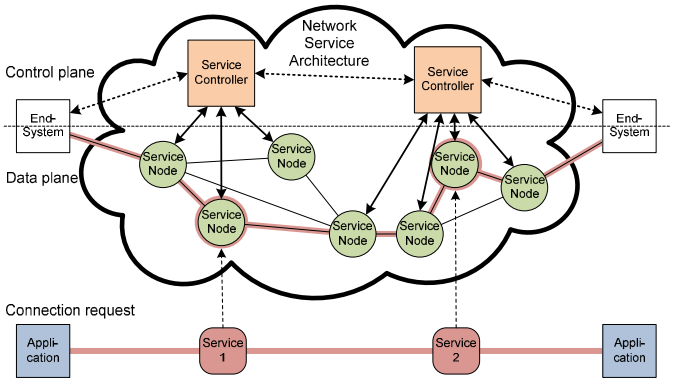
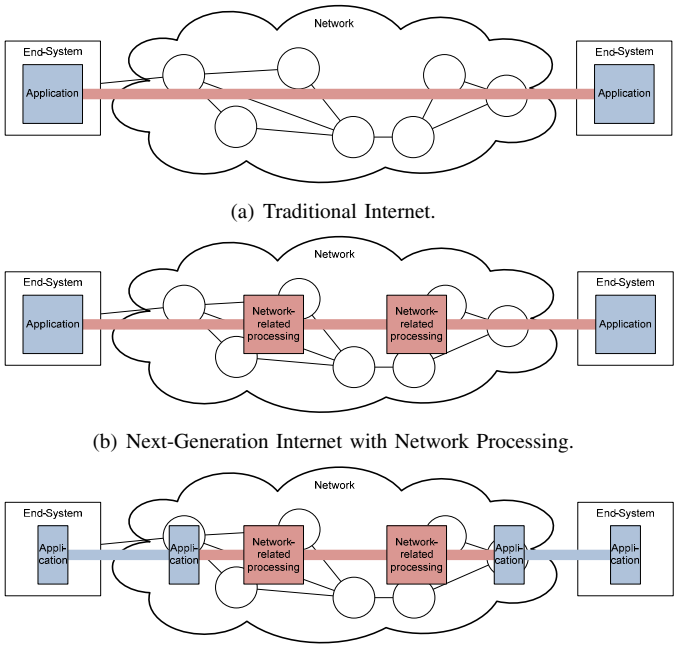


Fig. 2. Network Service Architecture (from [5]).



(a) Traditional Internet.

(b) Next-Generation Internet with Network Processing.

(c) Next-Generation Internet with Network Processing and Application Offloading.

Fig. 3. Placement of Processing Tasks on End-Systems and Network Systems for Different Network Architectures.

connection setup is handled, the service controllers manage processing on service nodes in their domain. Between service controllers, a distributed routing protocol is used to determine the best connection routing and service placement [19]. End-systems use a service socket interface to connect to service controllers and to provide the specification of the sequence of service that should be set up [20].

The key aspect of this architecture is that the number of processing functions is limited. We envision that the number of services is in the order of tens, and the deployment of new services is controlled globally (e.g., through the IETF). However, the number of service combinations is unlimited. End-systems may request any combination of services and thus can create novel communication methods and application uses.

D. Generalization for Application-Layer Processing

As discussed previously, it is desirable to also push application-layer processing into the data path of networks. Therefore, we need to extend the design of existing data-path processing architectures to support this functionality. In the specific case of the network service architecture (and for programmable networks in general), it is necessary to provide a method for applications to deploy processing functions into the network.

The general idea of this approach is shown in Figure 3. Figure 3(a) shows the traditional Internet architecture, where all processing is limited to end-systems. Our previously developed network service architecture is shown in Figure 3(b). The extension to include application-layer processing services is shown in Figure 3(c).

To implement application-layer services while remaining within the network service concept, it is necessary to provide a generic "application-layer service" (ALS). This service performs processing tasks that are determined by the end-system application at connection setup time (rather than picking from functions of existing services). The ALS can be implemented by providing a generic processing environment, similar to what has been developed in the active networking community [8]–[10]. The main difference, however, is that the ALS is part of the network service specification and thus can be controlled more carefully (e.g., placement of ALS can be limited to a few nodes close to the end-systems involved in the communication).

With ALS available in the system, the application can push any portion of its processing task into the network and thus offload portions of the application-layer processing tasks into the network.

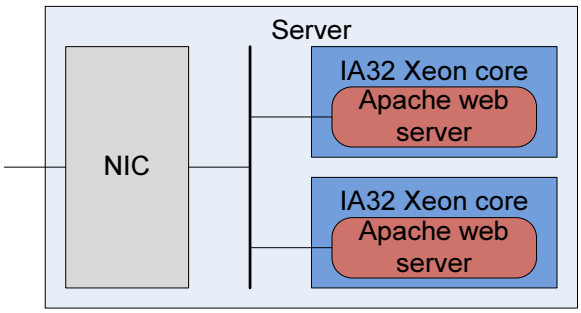
IV. Application Offloading: Web Server

With an understanding of how application-layer processing can be integrated into the network infrastructure, we turn towards a specific example. We discuss how a web server application can offload portions of its functionality into an application-specific network service.

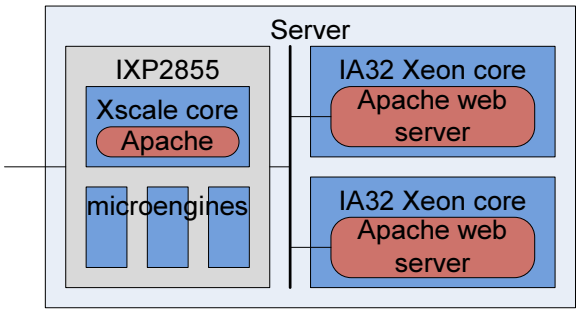
A. System Setup

In this example, we consider an Apache web server that serves both static and dynamic web pages. The end-system, where the Apache software is implemented is a server system with dual Intel Xeon cores. For data path processing, we use a Netronome NFE-i8000 system, which hosts an Intel IXP2855 network processor (NP) with one XScale control processor and sixteen data path processors [21]. There are three possible configurations on how this system can be used:

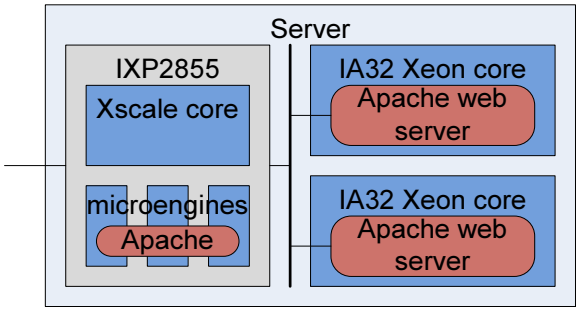
- Application-layer processing on end-system only: As shown in Figure 4(a), the system uses a conventional network interface card (NIC). The web server software runs in its entirety on the server processor cores.
- Application-layer processing on end-system and NP control processor: As shown in Figure 4(b), portions of



(a) Traditional Web Server Setup.



(b) Web Server with Offloading to Network Processor (Control Processor).



(c) Web Server with Offloading to Network Processor (Data Path Processors).

Fig. 4. Different Web Server Designs with Conventional Network Interface Card and Network Processors.

the Apache web server is offloaded to the NP's control processor.

- Application-layer processing on end-system and NP data path processors: As shown in Figure 4(c), the offloaded application-layer tasks are performed on the data path processors.

Conceptually, the second and third scenarios are equivalent since they both offload application-layer tasks into the network's data path. However, the implementation complexity (and performance) is different. Also, it is important to note that offloading is not limited to the same physical system as the end-system application. In our case, we co-locate the network processor and the end-system server for convenience. However, the application-layer processing could be performed anywhere in the network. The communication between the parts of the application use conventional network protocols (in our case TCP/IP).

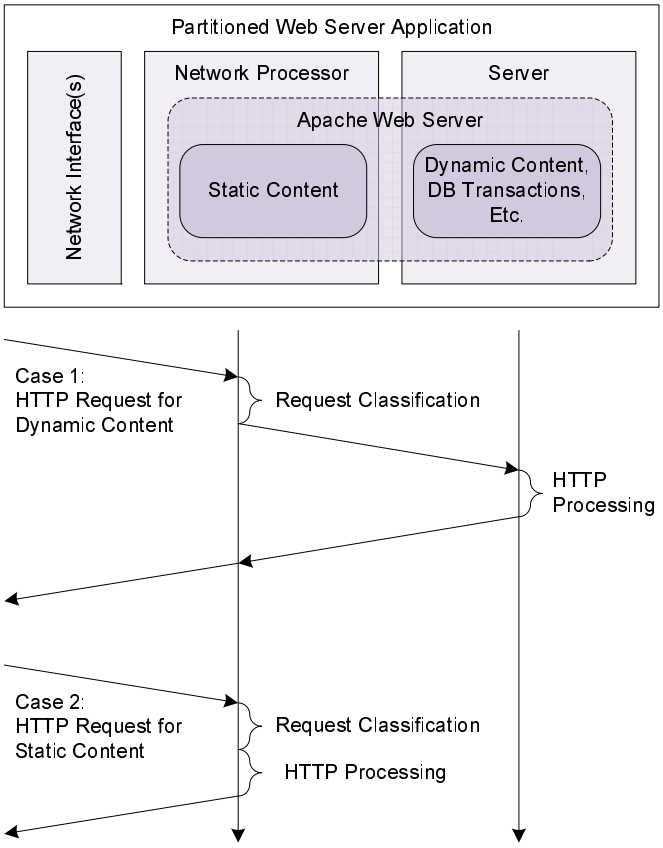


Fig. 5. Space-Time Diagram of Web Server and Offloading Operations.

B. Operation

One key question is which operations of the web server are performed inside the network and which are performed on the end-system. In our case, the in-network portion of the web server responds to static web pages. The end-system portion of the web server handles dynamic requests since they are considerably more complex (e.g., requiring database accesses).

The operation of the system is illustrated in Figure 5. The first case shows the handling of a request for a dynamic web page, which is forwarded to the end-system portion of the application. The second case shows how the offloaded portion of the server handles a request for a static web page.

C. Evaluation

We have prototyped two scenarios of web server system: one with all functionality on the end-system and one with offloaded static page handling in the data path (on the Xscale processor of the NFE-i8000). To classify between static and dynamic request, we implement a squid web proxy cache as the application-layer service. The cache can determine, which requests can be handled locally and which need to be passed on to the end-system. To exercise the system with a realistic workload, we use the SPECweb benchmark [22]. This benchmark generates parallel user session with request a mix of static and dynamic web pages (with sizes in the order of 10kB–100kB).

TABLE I
PROXY CACHE HIT RATE FOR SPECWEB BENCHMARK

Number of Sessions	Cache size 1MB	10MB	100MB
5	94.95%	95.05%	95.06%
50	92.71%	94.09%	95.03%
100	90.90%	93.10%	94.93%

TABLE II
ACCESS TIME SPEEDUP WITH OFFLOADING ON NETWORK PROCESSOR

Number of Sessions	Access time NIC	NP	Speedup
5	5.6s	5.6s	1.00×
50	6.7s	5.9s	1.12×
100	7.2s	6.8s	1.06×

One important question is how much memory is required for the application-layer service in the data path. Memory in network systems is often expensive and thus limited. Table I shows the cache hit rates for different numbers of sessions and cache sizes. It can be seen that as little as 1MB of memory provides effective caching, even for a large number of parallel sessions. Larger memories do not significantly improve the hit rates.

The speedup that is achieved for average access times to web pages is shown in Table II. We observe an improvement of up to 12% faster access times when using application-layer offloading. While this may not appear to be a significant change over the conventional system, it is important to note that our configuration assumes a worst-case placement of the ALS by co-locating it with the end-system application. The farther it can be pushed into the network, the sooner it can intercept requests and further reduce the access time. Since we do not have a large networking testbed, we cannot experimentally evaluate this scenario. However, the result in Table II confirms that a speedup is indeed possible and thus application-layer service in the network can improve application performance.

V. SUMMARY

We argue that applications can benefit from offloading processing tasks into the networking infrastructure. The use of network service abstractions allows a controlled deployment of application-layer processing. Our example of a web server application shows that offloading can improve the page access time and thus improve application performance. We believe that application-layer services are an important aspect for next-generation networks.

ACKNOWLEDGEMENTS

We would like to thank the Intel Embedded and Communications University Program for their support.

REFERENCES

[1] A. Feldmann, "Internet clean-slate design: what and why?" *SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 59–64, Jul. 2007.

[2] 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.

[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, "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.

[5] 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.

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

[7] F. Baker, "Requirements for IP version 4 routers," Network Working Group, RFC 1812, Jun. 1995.

[8] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, "The SwitchWare active network architecture," *IEEE Network Special Issue on Active and Controllable Networks*, vol. 12, no. 3, pp. 29–36, Aug. 1998.

[9] S. Choi, D. Decasper, J. DeHart, R. Keller, J. Lockwood, J. Turner, and T. Wolf, "Design of a flexible open platform for high performance active networks," in *Proc. of the 37th Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sep. 1999, pp. 157–165.

[10] S. Merugu, S. Bhattacharjee, E. W. Zegura, and K. Calvert, "Bowman: A node OS for active networks," in *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 1127–1136.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] *Maximizing HP StorageWorks NAS Performance and Efficiency with TCP/IP offload engine (TOE) Accelerated Adapters*, Hewlett-Packard Company, Mar. 2003, http://www.alacritech.com.

[17] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, and A. Foong, "TCP onloading for data center servers," *Computer*, vol. 37, no. 11, pp. 48–58, Nov. 2004.

[18] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "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, Jun. 2007, pp. 160–161.

[19] X. Huang, S. Ganapathy, and T. Wolf, "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, pp. 318–327.

[20] S. Shanbhag and T. Wolf, "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] *Intel Second Generation Network Processor*, Intel Corporation, 2005, http://www.intel.com/design/network/products/npfamily/.

[22] *SPECweb 2005 - Version 1.20*, Standard Performance Evaluation Corporation, http://www.spec.org/, Dec. 2007.