Design of a Network Service Architecture

Sivakumar Ganapathy and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA USA
{sganapat,wolf}@ecs.umass.edu

Abstract— **Considerable research efforts in the networking community are focused on defining a new Internet architecture that not only solves some of the problems of the current design, but also meets future needs. Our work focuses on the issue of how to provide suitable abstractions for communication between end-systems. This is a particularly important aspect of the network architecture as it is exposed to all applications and also determines what kind of services can be provided by the network. We present an architecture for network services that are implemented on router systems. We illustrate how control and data plane interact to provide the necessary end-to-end functionality. Our prototype implementation indicates that such a design is feasible and scalable for high-performance networks.**

I. Introduction and Related Work

The Internet has established itself as a successful and dominating example of a computer network. Careful considerations in the design philosophy of the Internet [1] have made it possible to develop a scalable and (mostly) decentralized system that connects hundreds of millions of systems. But when looking at recent and expected future technology developments, it becomes clear that some of the original assumptions and constraints in the Internet design no longer hold. Thus, the network research community is currently in the process of designing new network architectures that can address current and future challenges [2], [3] and eventually evolve or replace the current Internet.

In the context of a next-generation Internet architecture, many different proposals are considered. We can observe the following overarching themes:

- Increasing Diversity of End-Systems and Applications: The emergence of mobile thin-clients (e.g., cell phones, PDAs) and sensor networks have expanded the traditional view of workstation-type clients. Novel applications like peer-to-peer networks, multimedia streaming, and voice-over-IP have expanded the range of applications. Novel networking concepts that go beyond traditional store-and-forward approaches are introduced through content-distribution networks and caching architectures.
- Virtualization of Networking Hardware: The diversity in systems and applications makes it apparent that this diversity needs to be reflected in the networking infrastructure. Virtual networks that use "slices" of physical hardware have been proposed and prototyped [4], [5].
- Inherent Security Considerations: The implementation of security features in form of cryptographic processing on end-systems and firewalls and intrusion detection systems on the network edge is insufficient in today's environment. Several approaches attempt to develop a comprehensive security architecture that is inherent to the network rather than added afterwards [6], [7].
- Reconsideration of End-to-End Data Transfers: The design of the current Internet assumes most complexity of data communication to be placed in end-systems [8] and routers to be simple store-and-forward systems. New developments in network designs and applications make it necessary that routers perform more complex features (which we call "services"). Recent proposals suggest to consider these services as first-class networking features [9].

This theme of new abstractions for end-to-end communication that involve services inside the network is the topic of this paper. We address the question of how to design a network architecture where network services can be accessed explicitly by end-systems to utilize them for novel end-to-end communication paradigms.

Network services – while not formally part of the current Internet design – have been introduced slowly to address some shortcomings in the network design: A variety of so-called "middleboxes" are currently used to expand the IP address space (e.g., network address translation (NAT) [10]), provide security features (e.g., firewalls [11] and intrusion detection systems (IDS) [12]), and support different business models (e.g., ad insertion in HTTP traffic). These middleboxes are introduced into the network in such a way that they typically do not interfere with existing protocols. This requirement limits the features of middleboxes that can be deployed in such a fashion.

An alternative to stealthily adding middleboxes to a network is to make network services an explicit architectural feature. This expansion to store-process-forward networking has been approached in the past. Active networking proposed to give end-systems full access to program router systems [13]. The arising issues of security, resource management and isolation, and programming complexity limit the practicality of such a general approach to network services. A more constrained model of programmable routers is a more realistic scenario for deployment [14]. In programmable routers, a selected set of features as installed by an administrator and end-systems may choose if their communication utilizes these services.

The approach of explicit service features in the network also yields the benefit that such services can be visible to the

This work was supported by NSF grant CNS-0626690.

end-system and thus managed accordingly. The management issue of middleboxes has been addressed by the IETF Open Pluggable Edge Services (OPES) working group [15] and the IRTF End-Middle-End (EME) working group.

In our recent work, we have proposed a more fundamental shift towards network services, where services are not just optional add-on features, but *all* processing steps for data communication are considered a service [9]. These services encompass functions that have been traditionally placed on end-systems (e.g., information encoding, reliability protocols (e.g., TCP)) and functions that are typically placed on routers (e.g., NAT, IDS). This approach unifies all processing aspects of data communication into a common framework and thus provides a way for merging traditional networking abstractions with emerging communication paradigms.

The contributions of this paper are a specific design for a network service architecture that implements the concepts that are described only very generally in our prior work [9]. We present a discussion of design considerations for nodes that implement network service and for the control infrastructure that manages these services. Our prototype implementation and our performance results illustrate the feasibility of our design and the idea of end-to-end service abstractions in general.

The remainder of the paper revisits a more detailed discussion of network services in the end-to-end context in Section II. Section III presents the design of our architecture from an inter-network as well as an intra-network point of view. The prototype implementation is briefly discussed and performance results are presented in Section IV. Section V summarizes and concludes this paper.

II. Service-Centric End-to-End Architecture

To provide the necessary context for our system design, we briefly revisit the concepts of network services and how they are used for end-to-end communication as discussed in [9].

A. Concepts

The key idea for our end-to-end abstraction is to focus on the basic purpose of a network: *information transfer*. The encoding of information into data and the transfer of this data is a secondary step, and there are many ways of how this can be implemented. In our service-centric end-to-end architecture, end-system applications specify the information transfer that is desired and the network (in cooperation with the end-systems) determines the appropriate handling of data. There are three major aspects that need to be considered in this process:

- Data Encoding and Semantics: The information that needs to be transmitted needs to be encoded into data. In many cases, this is already given, for example in a file transfer.
- Data Transmission: The transfer of data between end-systems and network systems corresponds to the traditional functionality of a network.
- Data Processing: In order to provide advanced services, data needs to be processed. This includes modification of the transferred data and processing for the purpose of control.

The last point – having processing as a first-class networking function – is the key difference to the existing Internet architecture. Instead of limiting processing to end-systems, as it has been done in the current layered Internet architecture, we permit processing throughout the network. The purpose of these data services or network services is to implement the handling of data that is necessary to achieve the desired functionality and performance properties.

By putting the encoding of information under control of the network, we allow the system to maintain semantic information on the bits that are being transmitted. The network can therefore modify the data to more efficiently transfer the information. In the example of web caching, a proxy could intercept web requests and respond with a local copy of a document. As a result the same *information* is transferred without involving an end-to-end *data* transfer. This process would not require any configuration on the end-system since it could be done transparently by the network.

B. Network Services

Network services are key to providing flexible handling of data streams in the network. A service can receive, store, process, and transmit data that is sent over the network. Services leverage semantic information about a data stream to implement different functionalities. Examples for network services are:

- Reliability: buffering and acknowledgement-based retransmission of lost data as it is done in TCP.
- Privacy: encryption and decryption services between end-systems (similar to SSL) or subnets (similar to VPN).
- Congestion Control: limiting of data transfer rate based on the state of the (sub-)network.
- Caching: storing of information for certain application-layer protocols and making it accessible to other connections.
- Security: firewalling, intrusion detection, payload scanning and other mechanism to identify and mitigate attacks.
- Quality of Service: prioritized forwarding of data based on service requirements.
- Multicast: duplication and forwarding of data along multiple links and local retransmission for scalable, reliable multicast.
- Payload Transcoding: depending on the semantics of the transferred information, this service can adapt the content that is transferred. For example, large images from web pages can be downsampled for display on a cell phone. This service is very specialized and highly dependent on the transferred information and its coding in data.

Note that some of these services should be implemented on end-systems as traditionally done in the current Internet (e.g., reliability service). However, it is important that these services are also available inside the network since that allows routers

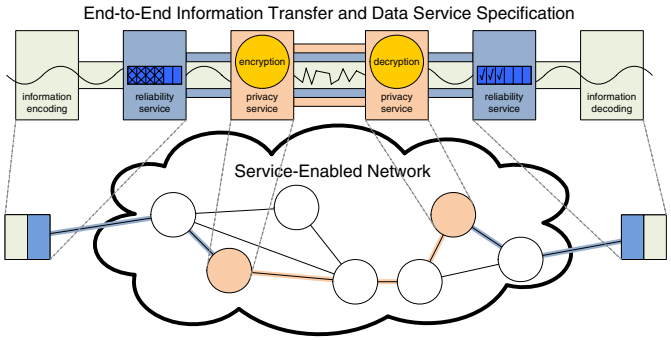


Fig. 1. Specification of End-to-End Information Transfer and Mapping of its Service Components onto Network Resources.

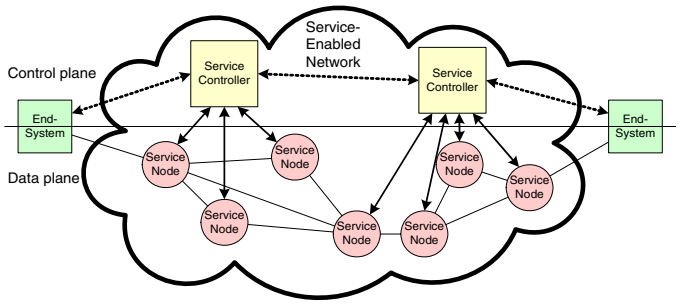


Fig. 2. Control and Data Path Components of Network Service Architecture Design.

to implement features that have been previously reserved for end-systems (e.g., the reliability service can be used to reassemble a TCP stream on an intrusion detection system).

C. Design Challenges

With these concepts of information transfer and network services, end-to-end communication can be specified by a sequence of processing tasks that need to be performed. This is illustrated in Figure 1 where one example of end-to-end communication is shown on top. The exchange involves reliability and privacy services that are mapped to service nodes in the network.

To implement a network service architecture, we need to address a number of fundamental theoretical and practical questions:

- How can nodes that implement services be managed in a scalable fashion?
- What information do end-systems, control nodes, and service nodes exchange?
- What per-flow state needs to be managed across components?
- What steps need to be taken for connection setup/teardown and data transfers?

We present an architecture design that addresses these questions and provides a practical environment to implement network services.

III. Design of Service Architecture

We split the discussion of our design into two aspects: first, we discuss how to design a scalable "inter-network" architecture that determines how services are set up across networks that are managed by separate administrative entities. Then we dive into the details of how services are managed and implemented within a network and a service node.

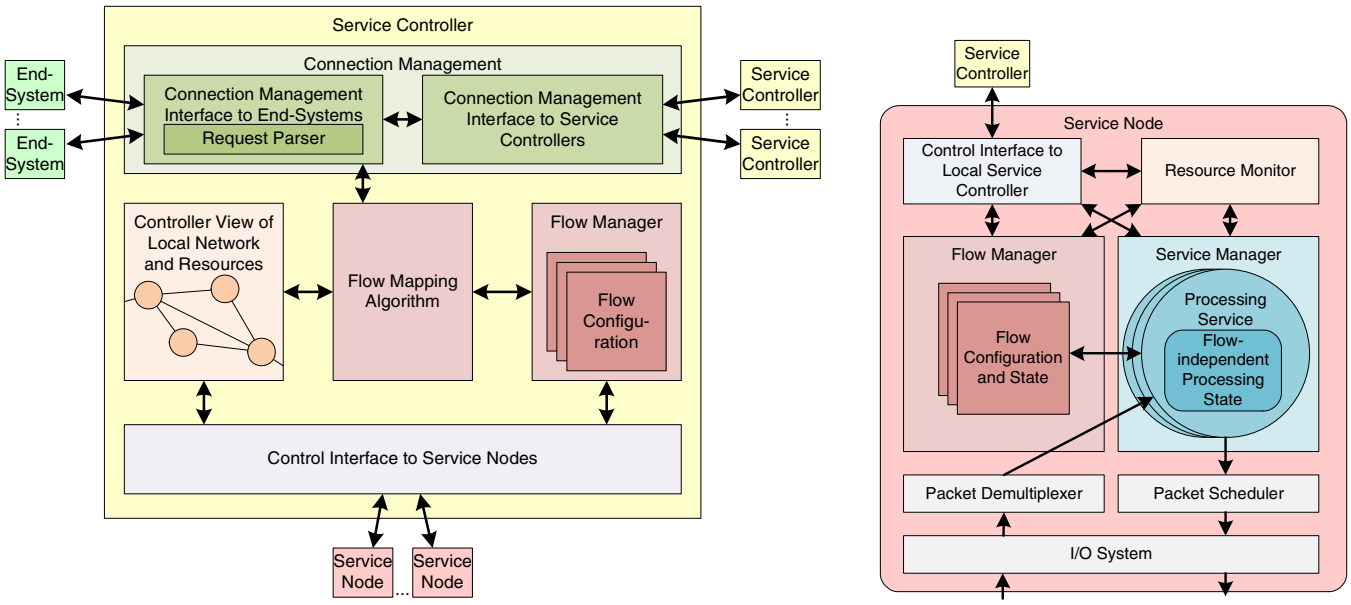
A. Inter-Network Design

The high-level design of our service-centric network is shown in Figure 2. The control plane is shown on the top of the figure and the data plane is shown on the bottom. When setting up a service-based communication, end-systems communicate with a "service controller" that manages a number of service nodes. This controller allocates processing services to service nodes that it manages and arranges the data transfers between them. For services that cannot be allocated to nodes that the controller manages (either due to resource limitations or due to placement constraints specified by the end-system), the request is passed on to a neighboring service controller that is along the path to the destination.

This design implies that a circuit-switched network with fixed routes for each flow is assumed. While the issue of state management in protocols is already a complex issue [16], it is particularly difficult in the context of services. Many processing services require that processing state is maintained between packets of the same flow (e.g., stateful packet inspection, measurement and monitoring, etc.). If routes change during the lifetime of a flow, then this state may not be available at another service node. It is conceivable that state may be moved from one service node to another during a rerouting event, but this is very difficult to implement and thus is not further considered here. For stateless services, a conventional packet-switched network can be assumed.

In a realistic deployment scenario, it can be expected that many complex and stateful services are implemented on or close to the end-system and the edge network (e.g., reliability service, content inspection, security services, etc.). In contrast, the performance demands in a core network limit the complexity of services to simple, stateless functions. This separation of complexity would allow a hybrid approach where fixed routes are required in the access and edge networks, but not in the core of the network. Thus, this design can be incrementally deployed in the existing Internet.

To set up services and routes, we envision a hierarchical approach as proposed previously in Private Network-to-Network Interface (PNNI) signalling for ATM networks [17]. As described above, service controllers control a number of service nodes and set up services among them. This presents one level of a hierarchy that can be further extended (e.g., service controllers that coordinate multiple other service controllers). During a connection request, services are then allocated among the highest level of controllers and then passed down the hierarchy to individual service nodes.



(a) Service Controller (b) Service Node

Fig. 3. Block Diagram of Design of Service Components.

B. Intra-Network Design

Within a network (i.e., the domain controlled by a single service controller), the service controller manages resource monitoring and connection setup and teardown. A detailed design of the service controller and service node is shown in Figure 3 and explained in more detail below.

1) Service Controller: The design of the service controller consists of four major components: connection management interfaces (between end-systems and other service controllers), resource management (tracking of available processing and memory on service nodes), flow-specific components (flow management, mapping), and control interface to service nodes. There are a number of design choices and considerations for each component.

a) Connection Management Interfaces: This component communicates with end-systems to receive requests for service-based communications. These requests can be specified in a number of different ways [18], [19]. We have chosen the "active pipe" abstraction proposed in our prior work. This specification notation provides a mechanism for expressing services, their parameters, and placement constraints in a textual string (for details see [9], [18]). This string is communicated to the service controller during connection setup. The request parser component analyzes the string and translates this into a representation that can be used by the mapping algorithm. The convenient aspect of the active pipe notation is that service components that are not handled by the service controller can be passed on to another service controller downstream.

b) Resource Management: The resource management component tracks the available resources on local service nodes. This information is necessary to determine if new service requests can be accepted. For our system, not only bandwidth needs to be tracked, but also availability of processing power and memory. When using heterogeneous systems, resources need to be normalized to a common metric as proposed in [20]. The simplest implementation of this component is "open-loop" where the available resources are estimated based on previous allocation. Nodes do not provide direct feedback on their actual state. Several problems can arise from this approach, in particular when resource requirements vary depending on changes in flow bandwidth, packet payloads, etc. A "closed-loop" implementation where nodes report periodically about their available processing, memory, and link resources is therefore preferable.

c) Flow Setup and Management: This component consists of the mapping algorithm for determining service placement and the flow manager. The mapping algorithm uses the service request translated by the request parser and attempts to map it to local service nodes. The information provided by the resource management components provides the constraints for potential mapping solutions. There are a number of approaches on how to perform mapping. The simplest case is a greedy approach or a load-balancing approach that only considers one metric (e.g., available processing power). If secondary metrics (e.g., memory and bandwidth) cannot be satisfied, the next node is considered. While this algorithm can be implemented easily, it is unsatisfactory to only consider one resource as the dominating metric. Instead, it is possible to use more advanced mapping algorithms to combine processing resources and bandwidth [21] or all three components [19].

d) Service Node Interface: This component communicates with the local service nodes that are managed by the service controller. It sends control messages for connection setup

and teardown and receives updates on available resources.

2) Service Node: The design of the service controller consists of five major components: control interface to the service controller, flow manager, service manager, resource monitor, and data I/O system.

a) Service Controller Interface: This interface communicates with the service controller to receive connection requests and sends updates on available resources. This is the complement component to the service node interface on the service controller.

b) Flow Manager: The flow manager not only keeps track of which flows are currently using service on the service node, but it also sets up the appropriate flow classification rules in the packet demultiplexer to ensure that each flow receives the appropriate service processing. For services that maintain state between packets across the flow, this state is maintained by the flow manager. In this context, it is important to point out that this design separates the state of per-flow data structures (e.g., encryption keys) from the state of per-service data structures (IDS rules).

c) Service Manager: This component manages the actually processing that implements network services. This requires that processing resources are made available to the packets that are passed from the demultiplexer. Also, flow-independent per-service state is managed by the service manager.

d) Resource Monitor: The resource monitor monitors the operation of the service manager. It reports the available resources to the service controller. If resources are abused or locked up due to erroneous behavior, it can reset services and free up processing resources and memory. This of course may impact per-flow state and thus needs to be coordinated with the service controller.

e) Data I/O System: The I/O system receives and transmits the actual packets that are forwarded and processed by the service node. As indicated above, the packet demultiplexer ensures that packets are passed to the correct service component (or simply forwarded if there is no service to be performed). The packet scheduler is used to ensure that the access to the outgoing link can be controlled by the system. Since some services send more data than they receive (e.g., multicast, decompression) the outgoing link can become overloaded.

If this design is used on a larger multi-port switch, it is possible to place the service controller on the control processor of the router. Each I/O port can then be equipped with a service node that runs on a port processor or network processor.

IV. Prototype Implementation

We have implemented a prototype of the network-service architecture that uses the concepts discussed above. This section presents an overview on this implementation and a few performance results to illustrate its operation.

A. Prototype Setup

In our prototype, a service controller is placed on a well-known host. Each service node that is started initially connects to the service controller and "registers" its available service. This information is kept by the service controller in the resource management component that provides a local view to the mapping algorithm. Our prototype system currently supports three services: reliability (using a simple stop-and-wait protocol), compression/decompression, and encryption/decryption.

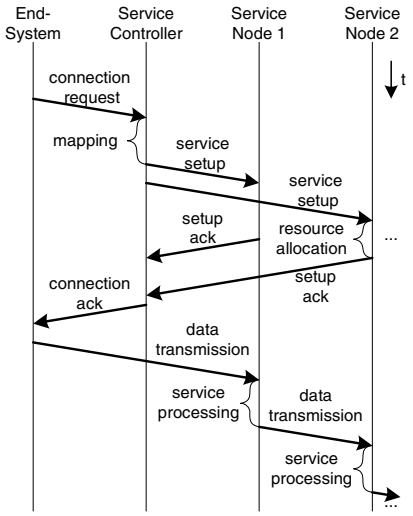


Fig. 4. Example of Connection Setup and Data Transfer Interactions between End-System, Service Controller, and Service Node.

When an end-system wants to use network services, it sends a request to the service controller. The controller parses the request string and determines which nodes should implement the requested services. These nodes are then notified and hop-by-hop UDP connections are established. That is, each node is informed to which IP address and port number to send the packet next. After setting up the service nodes, the end-system is informed where the first hop of the connection is and to which port to send traffic. All traffic is sent via UDP since all current services are implemented in user space. In the next generation of our prototype, we envision services to be implemented in the kernel. This would allow us to use a custom demultiplexing function rather than relying on UDP and its port numbers.

An example of the operation of the prototype is shown in the space-time diagram in Figure 4. In this scenario, the end-system requests a connection that uses multiple network services. The service controller performs a mapping process to determine where to place which services. It then communicates with two service nodes to set up the required processing resources and communication links. Once the service nodes have acknowledged the connection setup, the service controller communicates back to the end-system. After this connection setup process, data is transmitted and processed on the service nodes (without further involvement of the service controller). Note that only one end-system is shown.

B. Prototype Performance Results

To give an insight in the performance of our proposed design, we have measured the connection setup delay for a

Step	Time
Connection request, parsing, mapping	475 μs
Service setup and resource allocation	463 μs
Total connection setup	938 μs

TABLE I
AVERAGE CONNECTION SETUP PERFORMANCE.

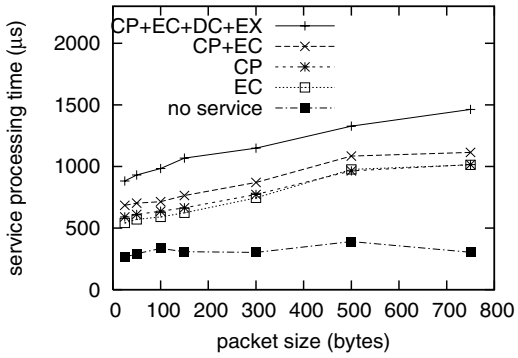


Fig. 5. Service Node Processing Performance for Different Combinations of Services (CP/EX=compression/expansion, EC/DC=encryption/decryption).

service request as well as the service processing performance. The results are obtained from our prototype implementation, where the service controller and service nodes are implemented on a workstation system (Intel x86, 3.0GHz, Linux 2.6.15 kernel).

Table I shows the setup time for a connection which involves four network service steps that are placed on two service nodes. The processing of the request on the service controller node requires 475 μs and the service setup requires 462 μs. Thus, the overall delay from the point of view of the end-system is roughly 1 ms.

The service processing time for different configuration is shown Figure 5. The baseline forwarding performance is illustrated with the configuration that uses no services. In that case, packets are simply passed from the input interface to the output interface. For the other configurations, we use the encryption and compression services, which process packet payloads. Thus, there is an (approximately linear) increase in processing time with increasing packet size. When going from one to two services, the processing time does not double, but increases by less. This is due to a fixed per-packet overhead in the service node.

V. SUMMARY AND CONCLUSIONS

We have presented a design for a network architecture that improves on existing end-to-end abstractions by providing processing service inside the network. We have reviewed the general design concepts of such an architecture and the discussed a detailed design. We have illustrated how the service controller and service node of our architecture operate. Our prototype implementation is discussed and the performance results show that the proposed design can indeed be implemented and operate efficiently. We believe that this work provides an important step towards developing a next-generation Internet that overcomes the limitations of current network designs.

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] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the internet: the end-to-end arguments vs. the brave new world," *ACM Transactions on Internet Technology*, vol. 1, no. 1, pp. 70–109, 2001.

[3] D. Clark, K. Sollins, J. Wroclawski, D. Katabi, J. Kulik, X. Yang, B. Braden, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa, "New Arch: future generation internet architecture," Defense Advanced Research Project Agency, Tech. Rep., Dec. 2003.

[4] 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.

[5] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, Aug. 2006, pp. 3–14.

[6] S. M. Bellovin, D. D. Clark, A. Perrig, and D. Song, "A clean-slate design for the next-generation secure internet," National Science Foundation, Tech. Rep., Mar. 2005.

[7] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. B. N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," in *15th USENIX Security Symposium*, Vancouver, Canada, Aug. 2006, pp. 137–151.

[8] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, Nov. 1984.

[9] 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.

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

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

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

[13] 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.

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

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

[16] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A comparison of hard-state and soft-state signaling protocols," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, Aug. 2003, pp. 251–262.

[17] *Private Network-Network Interface Specification Version 1.0*, ATM Forum Technical Committee, Mar. 1996.

[18] R. Keller, J. Ramamirtham, T. Wolf, and B. Plattner, "Active pipes: Program composition for programmable networks," in *Proc. of the 2001 IEEE Conference on Military Communications (MILCOM)*, McLean, VA, Oct. 2001, pp. 962–966.

[19] L. Ruf, T. Wolf, K. Farkas, and B. Plattner, "Specification of network services and mapping algorithms," in *Proc. of the 2006 IEEE Conference on Military Communications (MILCOM)*, Washington, DC, Oct. 2006.

[20] V. Galtier, K. L. Mills, Y. Carlinet, S. Leigh, and A. Rukhin, "Expressing meaningful processing requirements among heterogeneous nodes in an active network," in *Proc. of the Second International Workshop on Software and Performance*, Ottawa, Canada, Sept. 2000, pp. 20–28.

[21] S. Y. Choi, J. S. Turner, and T. Wolf, "Configuring sessions in programmable networks," in *Proc. of the Twentieth IEEE Conference on Computer Communications (INFOCOM)*, Anchorage, AK, Apr. 2001, pp. 60–66.