NetServ: Active Networking 2.0

Jae Woo Lee*, Roberto Francescangeli†, Jan Janak‡, Suman Srinivasan*, Salman A. Baset*,
Henning Schulzrinne*, Zoran Despotovic§ and Wolfgang Kellerer§

* *Department of Computer Science, Columbia University, New York, USA*
{jae,sumans,salman,hgs}@cs.columbia.edu
† *DIEI, Universita degli Studi di Perugia, Italy*
roberto.francescangeli@diei.unipg.it
‡ *Tekelec, North Carolina, USA*
jan@iptel.org
§ *DoCoMo Communications Laboratories Europe, Munich, Germany*
{despotovic,kellerer}@docomolab-euro.com

***Abstract*—We present NetServ, a node architecture for deploying in-network services in the next generation Internet. NetServ-enabled network nodes provide a common execution environment, where network services implemented as modules can be dynamically installed and removed.**

We demonstrate three such modules. MicroCDN is a dynamic content distribution network (CDN) service which implements a content caching strategy specific to a content provider. The NAT Keep-alive module offloads the processing of keep-alive messages from SIP servers. The Media Relay module allows any NetServ node to act as a media relay, eliminating the need to manage standalone relay servers.

NetServ aims to revive the Active Networking vision. It was too far ahead of its time a decade ago, but we believe its time has finally arrived.

I. Introduction

The architectural simplicity of the core Internet is a double-edged sword. On the one hand, its agnostic nature paved the way for endless innovations of end-to-end applications. On the other hand, the inherent limitation of this simplicity makes it difficult to add new functions to the network core itself. This is exacerbated by the conservative tendency of commercial entities to "leave well-enough alone", leading to the current situation often referred to as the *ossification* of the Internet. For decades, there has been practically no new functionality that has been added to the core Internet on a large scale. When there was a need, researchers and entrepreneurs had to find a clever roundabout ways to implement the desired functionality in the application layer rather than in the network layer even though the latter would have led to a superior solution.

This is changing now. Many researchers believe that it is no longer possible to *patch up* the Internet in the face of exploding bandwidth usage and ever-increasing demand for new in-network functionalities. Bold proposals that advocate clean-slate redesigns of the Internet architecture are no longer perceived as merely satisfying academic curiosity. In fact, the National Science Foundation of the United States has launched GENI [1], a large-scale project involving academic and industrial teams across the country, which aims to provide an Internet-scale testbed for such disruptive proposals that cannot be tested on the current Internet.

In this paper, we address the question of enabling in-network functionalities in the next generation Internet. We start by assuming that two fundamental characteristics of today's Internet will survive in the future: the packet transport (think IP today) and the concept of addressable services (think web servers today.) We jettison, however, the dichotomy that the two characteristics impose on the Internet actors today. On the one hand, ISPs operate routers which process packets in the network core–forwarding, monitoring and manipulating them–but routers do not normally provide addressable services. On the other hand, service providers and end users operate host computers which provide addressable services, but the providers and users do not usually have a means to process incoming packets in the network core. What if service providers can deploy network services on a backbone router? What if end users can deploy a packet monitoring tool on their ISPs' edge routers?

We present NetServ, a common node architecture to realize this vision. Network services and functionalities are implemented as software modules which can be deployed at any NetServ node on the Internet, subject to policy restrictions. The NetServ nodes provide a common execution environment for the service modules, so that they can be easily moved around among the nodes. Our ultimate vision is to provide a common API, virtualized execution environments, and a signaling protocol, so that a network service can be freely installed, removed, and migrated among the Internet nodes of all kinds–from a backbone router to a set-top box at home.

We describe the NetServ architecture and our prototype implementations in Section III. In Section IV, we present three NetServ applications that we are developing: MicroCDN, Media Relay, and NAT Keep-alive responder. We compare NetServ with earlier work by others in Section V. But first, we must explain why we regard NetServ as Active Networking 2.0.

978-1-61284-955-3/11/$26.00 ©2011 IEEE

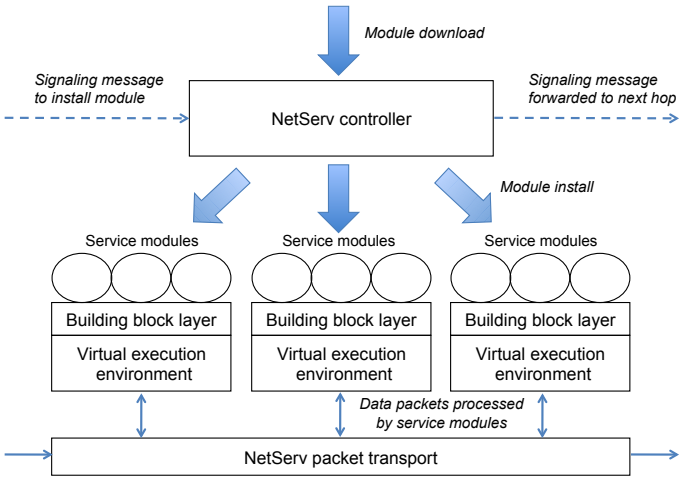

Fig. 1. General NetServ architecture.

II. Active Networking Redux

The idea to make the network elements programmable is not new. Active networking articulated the same vision more than a decade ago [2]. In fact, active networks went even further, advocating the *integrated* approach, where every packet can carry a program that can alter the behavior of network nodes. Many researchers attribute the ultimate demise of active networks to the security risk associated with user packets carrying code–or at least to the perception of that risk.

NetServ can be viewed as a revival of active networks. We put forth essentially the same vision. However, there are a few reasons why we believe that NetServ will yield a very different outcome than that of active networks. First, among the two approaches to programmability that active networks suggested, we forgo the integrated approach and adopt the more conservative *discrete* approach, where network elements are programmed by out-of-band signaling. Second, advances in the area of virtualization and isolation have been considerable in the past decade. The use of virtual machines is widespread across many applications. Java-based sandboxing mechanisms have matured over the years. Third, we note the industry trend to provide programming APIs for 3rd party developers to extend their products by the router vendors who have been very resistant to open up their technology in the past [3]. Last but not least, we believe the timing is right. The increasingly evident shortcomings of the current Internet architecture, and the urgency to fix it, will lead many researchers and industry participants to look at new (and old) ideas with an open mind.

III. NetServ Architecture

A. Overview

Figure 1 depicts the general NetServ architecture. The service modules, represented as ovals, are the applications that provide network services. They are written in Java and conform to the OSGi [4] component specification. The OSGi framework provides dynamic module installation and removal, and isolation between modules.

The service modules run in a virtualized execution environment. We envision three different types of execution environment providing different levels of isolation between modules, depending on the capability and policy of the NetServ node hosting the environment:

1) **Namespace isolation**: The modules run in a single Java Virtual Machine (JVM). The OSGi framework provides isolation between modules by loading them into separate namespaces, but it is hard to control resource consumption, such as CPU time spent by the modules.
2) **Process isolation**: The modules can be grouped into multiple JVM processes, one for each user, for example. The traditional process-level protection of multi-user operating systems can be used to provide resource control and isolation between the JVMs.
3) **Virtual machine isolation**: Further isolation and protection can be achieved through virtualization. For example, a group of modules running in a virtual machine can have their own IP address, allowing modules to become network server applications.

The execution environments communicate with the NetServ packet transport layer so that data packet flows can be routed to the appropriate service modules.

There are three types of modules: server module, packet processing module, and monitoring module. Server modules are network server applications that are explicitly addressable, such as a web sever module that listens on port 80 on a dedicated IP address. Packet processing modules are placed on the packet path, and process the packets transiting the NetServ node, which is normally a router in this case. Monitoring modules are similar to packet processing modules in that they also inspect transit packets, but they do not modify them. In addition, a NetServ node can impose a maximum rate at which packets are sampled for monitoring modules. Different NetServ nodes will support different types of modules. A NetServ end-user host might only support server modules and a NetServ core router might only support monitoring modules, while a NetServ edge router might support all three types of modules.

The NetServ controller is responsible for downloading a module and installing it into an execution environment. The controller responds to signaling messages sent by the neighboring NetServ nodes. The NetServ signaling protocol, based on NSIS [5], is used to install and remove modules, and to query the status of NetServ nodes. A NetServ node will usually require authentication before it accepts a module install request, but it may allow anonymous modules to run in an execution environment with a limited set of resources. The capability and policy of a NetServ node can be probed using the signaling protocol. A NetServ node can query its neighbor to discover, for example, the types of execution environment it supports, the types of module that can be installed, and whether authentication is required.

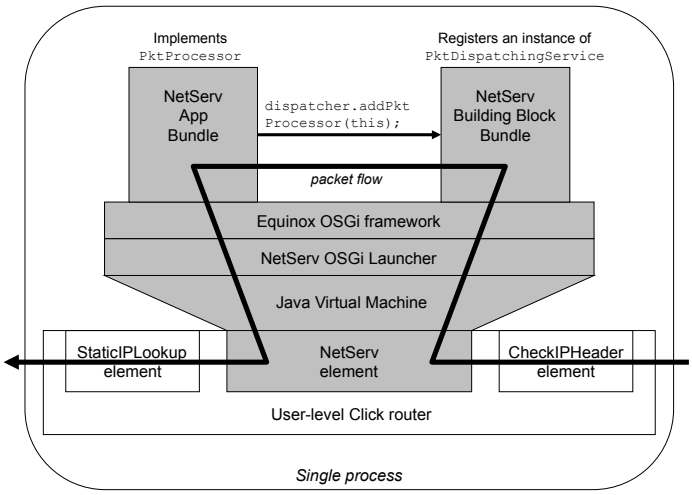

Fig. 2. First NetServ prototype.

B. Signaling

In the NetServ architecture, module installation and removal are triggered by the exchange of signaling messages. The NetServ signaling protocol extends the NSIS Framework protocol suite. The NSIS Framework describes a two-layer stack for signaling on the Internet. The lower layer, the NSIS Transport Layer Protocol (NTLP), is in charge of routing and transporting higher layer messages to the next NSIS node on the path. The higher layer, the NSIS Signaling Layer Protocol (NSLP), handles the actual signaling logic specific to a given application. The NTLP layer can also exchange control and feedback information, such as errors and route change indications, with the signaling layer.

The General Internet Signaling Transport (GIST) [6] is a concrete implementation of the NTLP. GIST handles application-independent signaling tasks related to message routing, security, peer discovery and connection state management. A NetServ NSLP daemon runs on top of the GIST daemon, and handles NetServ-specific signaling messages. The NetServ NSLP daemon registers itself with the GIST daemon, and the GIST daemon passes NetServ signaling messages up to the NSLP layer. A NetServ message received by the GIST daemon (and in turn by the NSLP daemon) can be addressed directly to the node, or it can be intercepted by an intermediate node along the path to the destination (path-coupled signaling).

C. Packet Processing

In our first prototype implementation, we explored one particular aspect of the NetServ architecture, namely, dynamic installation of packet processing modules. Figure 2 describes the implementation. We used the user-level Click router [7] as the packet transport layer, and the Java Native Interface as the communication channel between the packet transport and the OSGi module environment. Our previous paper [8] describes this implementation in detail and provides performance evaluation.

We are currently engaged in another prototyping stage, where we are building a more complete system outlined in Section III-A, including multiple execution environments, signaling, and the NetServ controller daemon.

For the current prototype, we have replaced the user-level Click router with two different kernel-level packet transport layers, which we are developing in parallel: the Linux kernel using Netfilter queue [9] as kernel-user communication, and the kernel-level Click router using pseudo-devices. We are also planning to build the same system on top of a Juniper router using the recently published JUNOS SDK [3].

D. Security

The main objective of the NetServ framework is to allow service providers install services (in form of Java modules) on Internet routers. Such routers may be running within other autonomous systems, i.e., autonomous systems that are in no relationship with the service provider. Due to the lack of relationship between the owner of the NetServ-enabled router and the service provider wishing to install a module on that router, the owner of the NetServ-enabled router would need to take precautions before installing the module and verify:

- The authenticity and integrity of the module being installed.
- The authorization of the module to access the advertised network prefix (specified by a matching expression inside the module).

The NetServ framework relies on standard Java 2 security mechanisms to verify the authenticity and integrity of modules being installed. In order to get access to the services provided by the NetServ framework on the router, modules being installed need to carry a signature of a trusted service provider. Without a valid signature the NetServ framework refuses to install the module.

Accounting and resource control mechanisms in the NetServ framework are used to keep track of the resources each service modules has used. This allows the service provider to be charged for resources their modules used, or to curtail operation of a module if it has exceeded its resource limits.

Fine-grained access control to networking resources is of utmost importance in the NetServ framework. The core of the NetServ framework needs to make sure that modules running on the router can only get access to the traffic that "belongs" to them and not to other network traffic passing via the router.

Each service module declares its interest in network traffic by providing a matching expression. The NetServ framework uses the matching expression to filter traffic passing through the router; only traffic that matches the expression will be passed onto the service module. Because matching expressions advertised by modules may contain arbitrary IP addresses or network prefixes, the NetServ framework also has to verify that the module is authorized to receive the traffic specified by its matching expression. In other words, the NetServ framework needs to ensure that modules cannot spoof traffic of other service providers.

We realize that our requirement to verify the ownership of a network prefix is very similar to the problem being solved in the Secure Inter-Domain Routing working group in the IETF. The working group proposes a PKI (Public Key Infrastructure) based solution called RPKI. The solution can be used to verify whether a certain autonomous system is allowed to advertise a given network prefix. We plan on using that infrastructure, once available, in NetServ. By mapping service providers to autonomous systems we can use the RPKI infrastructure to verify the right of a module to receive traffic to and from a set of IP addresses.

Because access to the RPKI infrastructure may not be available on all NetServ-enabled routers, NetServ will also support simpler verification mechanisms that do not rely on PKI, such as reverse routability checks and DNS based verification. The level of security verification and combination of security checks used by a NetServ-enabled router is a matter of local configuration policy and will be determined by the owner of the router.

IV. NetServ Applications

In this section, we describe three NetServ applications. MicroCDN is a dynamic Content Distribution Network (CDN) service which implements a content caching strategy specific to a content provider. The NAT Keep-alive responder and the Media Relay modules solve two common NAT-related problems in IP telephony. The NAT Keep-alive module offloads the processing of keep-alive messages from the SIP server. The Media Relay module allows any NetServ node to act as a media relay, thereby eliminating the need to manage standalone relay servers.

A. MicroCDN

The Internet has seen an explosive growth in traffic carrying multimedia content in recent years. To cope with such demand, larger content providers usually employ content distribution network (CDN) operators. The CDNs typically deploy a large number of content servers in diverse geographic locations, and use centrally controlled redirection mechanisms, such as DNS redirection, to redirect user requests to content servers closer to the users. The static nature of the redirection mechanism makes it somewhat difficult for the content provider and the CDN operator to adapt quickly to changing traffic patterns.

MicroCDN, a NetServ-based CDN application, aims to address the shortcomings of the traditional CDN architecture. Figure 3 shows how MicroCDN works in a simple scenario where users are downloading a video file from a content provider, YouTube.com. The content provider develops a MicroCDN NetServ module and makes it available for download. The content provider's server machines respond to the content requests as usual.

In addition, however, when a server notices the rate of requests from a network location above a certain threshold, the server initiates on-path signaling to tell the NetServ nodes in the path that they should download and install the provider's MicroCDN module. The signaling message contains the URL

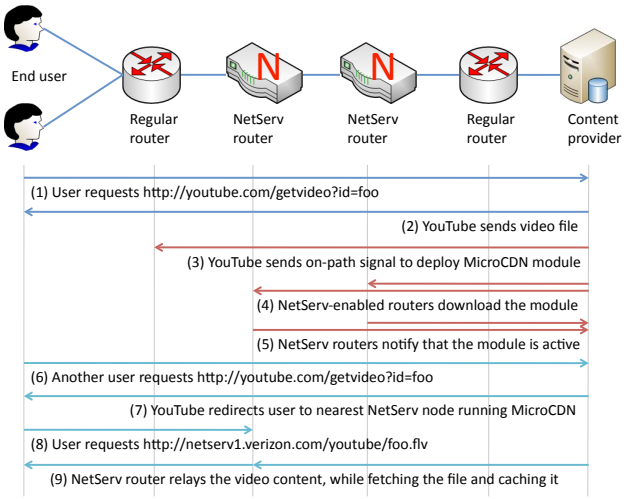

Fig. 3. On-demand content caching using MicroCDN.

and other information about the module, so that the NetServ nodes can determine if the module is compatible with the node's policy and capability, and where to download it from. After a NetServ node successfully downloads the module, it signals back the provider's server to register itself as one of the caching nodes. Now, the content requests originating from the vicinity of that caching node can be redirected to the node. The caching node will then fetch the requested content, sending it to the user as it is being fetched, and cache it for future requests.

The MicroCDN module will uninstall itself after a period of inactivity. And, of course, it can be reinstalled when new demands arise. The content provider controls the tunable parameters such as the inactivity time before module expiration. In fact, since the MicroCDN module is written by the content provider specifically for the provider, the provider controls every aspect of the module's behavior, from the cache replacement policy to the algorithm to locate the nearest caching node. The module can even modify the video content on the fly, inserting watermarks or advertisements, for example. This is indeed the biggest advantage of MicroCDN compared to the traditional content distribution network. Using MicroCDN, content providers can employ any distribution strategy that satisfies their need, rather than being locked in by the mostly static infrastructures of traditional CDN providers.

B. NAT Keep-alive responder

Unfortunately, NATs are part of the Internet fabric today. Internet Telephony Service Providers (ITSPs) are acutely aware of the effect of ubiquitous NAT boxes when they have to deploy a SIP server for a large number of subscribers. NAT boxes maintain an ephemeral state between internal and external IP addresses and ports, referred to as a binding. After a SIP User Agent (UA) behind a NAT box registers its IP address with the SIP server, the UA must send a periodic keep-alive message in order to prevent the NAT box from removing the binding. Without the keep-alive messages to refresh the binding, the UA will not be able to receive incoming calls.

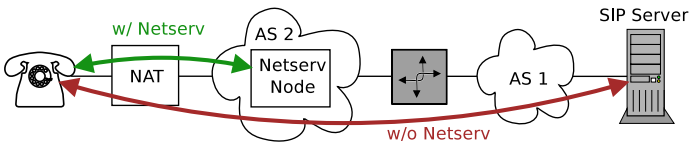

Fig. 4. Operation of NetServ NAT keep-alive responder.

Most UAs use UDP for SIP signaling, and reliably keeping a UDP binding in a typical NAT box requires that a keep-alive message should be sent every 15 seconds [10]. While the size of a keep-alive message is relatively small (about 500 bytes when SIP messages are used for this purpose, which is often the case), millions of users sending it every 15 seconds represents a significant consumption of network and server resources. This traffic wastes energy, adds to the operating cost of ITSPs, and serves no useful purpose–other than to fix a problem that should not exist in the first place. A surprising fact is that the keep-alive traffic is a major bottleneck in scaling a SIP server to a large number of users [10].

Figure 4 shows how NetServ could help to offload NAT keep-alive traffic from the ITSP's infrastructure. Without the NetServ keep-alive responder, the SIP UA behind NAT sends a keep-alive request to the SIP server every 15 seconds and the SIP server sends a response back. The NAT keep-alive packets can be either short 4-byte packets or full SIP messages. For our implementation, we are using full SIP messages because, to the best of our knowledge, this is what most ITSPs use.

When an NSIS-enabled SIP server starts receiving NAT keep-alive traffic from a SIP UA, it initiates NSIS signaling in order to find a NetServ router along the network path to the SIP UA. If a NetServ router is found, the router downloads and installs the NAT keep-alive responder module. Such a module would be typically be provided by the ITSP running the SIP server.

After the module has been successfully installed, it starts inspecting SIP signaling traffic going through the router towards the SIP server. If the module finds a NAT keep-alive request, it generates a reply on behalf of the SIP server, sends it to the SIP UA and discards the original request. Thus, if there is a NetServ router close to the SIP UA, the NAT keep-alive traffic never reaches the network or the servers of the ITSP; the keep-alive traffic remains local in the network close to the SIP UA.

The NAT keep-alive responder spoofs the IP address of the SIP server in response packets sent to the UA. This is necessary for certain restrictive NATs. IP address spoofing is not an issue in this case because the NetServ router is on the path between the spoofed IP address and the UA.

C. Media Relay

NAT boxes may also prevent SIP UAs from directly exchanging media packets, such as voice or video. This means that ITSPs must deploy media relay servers to facilitate the packet exchange between UAs behind NATs. However, this approach has several drawbacks, such as increased packet

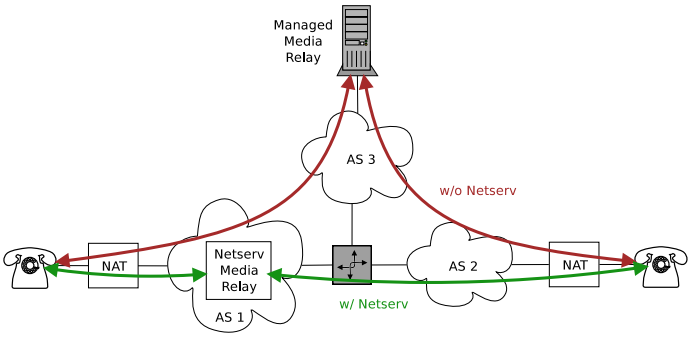

Fig. 5. Operation of NetServ media relay.

delay, additional hardware and network costs, and management overhead. One way to address these drawbacks is to deploy the media relay functionality at the edge of the network, on routers and hosts that are closer to UAs.

Figure 5 shows how NetServ helps to offload the media relay functionality from an ITSP's infrastructure. The direct exchange of media packets between the two UAs in the picture is not possible. Without NetServ the ITSP would need to provide a managed media relay server. When a NetServ router is available close to a UA, the SIP server can deploy the media relay module at the NetServ node.

When a UA registers its network address with the SIP server, the SIP server sends an NSIS signaling message towards the UA, instructing the NetServ routers along the path to download and install the media relay module. The SIP server then selects a NetServ node close to the UA, instead of a managed server, to relay calls to and from that UA.

D. Reverse data path

The previous descriptions of the three applications assumed that the reverse data path is the same as the forward path. On the Internet today, however, this is often not the case due to policy routing.

For MicroCDN and Media Relay, this is not an issue. The modules only need to be deployed *closer* to the users, not necessarily on the forward data path. The module will still be effective if the network path from the user to the NetServ node has a lower cost than the path from the user to the server.

For NAT Keep-alive responder, the module must be on-path to carry out its function. However, depending on the network topology of users and NetServ routers, this may not be a serious problem. If a module is installed at a place where there is no user to serve, it will time-out quickly. Furthermore, if we assume a dense population of users, it is likely that a module will serve some users, albeit not the ones who triggered the installation in the first place.

If on-path installation is indeed required, the client-side software can initiate the signaling instead of the server. In the NAT Keep-alive case, this means a modification to the SIP User Agent software. Another way to address this problem is to employ a round-trip signaling between the server and the last NetServ node on path.

V. Related work

NetServ can be regarded as a continuation of the body of active router work such as Router Plugins [11], LARA++ [12], PromethOS [13] and Pronto [14]. They define extensible router architectures for implementing new router features. The focus here is to enhance the routers' packet processing capabilities. Packet processing is a core part of NetServ as well. However, it is also a part of a bigger picture in NetServ, namely, making the network core a distributed service execution environment. NetServ provides a common framework for both packet processing and server-style applications. Our ultimate vision is for all nodes on the Internet, not just routers, to become NetServ nodes.

Our design choices clearly reflect this goal. The network services are implemented as user-level modules rather than kernel extensions. We consider the dynamic deployability of service modules as a critical requirement of the system. Java's portability ensures that modules can be installed anywhere, and NSIS signaling provides a mechanism for on-path installation, which does not require the discovery of specific NetServ nodes. Security and resource isolation remain as the primary objective in our current and future development.

VI. Conclusion

In this paper, we presented NetServ, a node architecture for deploying in-network services in the next generation Internet. NetServ-enabled network nodes provide a common execution environment, where network services implemented as modules can be dynamically installed and removed. We demonstrated three such modules: MicroCDN, NAT Keep-alive responder and Media Relay.

NetServ can be viewed as a revival of active networks. There are several reasons why we believe NetServ will successfully implement the Active Networking vision which has been dormant for a decade. First, we take a more conservative *discrete* approach, where network elements are programmed by out-of-band signaling. Second, we take advantage of the recent advances in virtualization and isolation technologies. Lastly, we believe it is the right time for a bold architectural initiative, given the increasingly evident shortcomings of the current Internet architecture.

VII. Acknowledgments

The NetServ project is funded by the National Science Foundation under grant NSF-CNS #0831912 as a part of its Future Internet Design (FIND) initiative, and also by DOCOMO Communications Laboratories Europe.

References

[1] NSF GENI. http://www.geni.net/.

[2] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 2, pp. 5–17, 1996.

[3] J. Kelly, W. Araujo, and K. Banerjee, "Rapid service creation using the JUNOS SDK," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 56–60, 2010.

[4] OSGi Alliance. http://www.osgi.org/.

[5] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, "Next Steps in Signaling (NSIS): Framework," RFC 4080, June 2005, http://tools.ietf.org/html/rfc4080.

[6] H. Schulzrinne and R. Hancock, "Gist: General internet signalling transport," IETF draft, June 2009, http://tools.ietf.org/html/draft-ietf-nsis-ntlp-20.

[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, August 2000.

[8] S. R. Srinivasan, J. W. Lee, E. Liu, M. Kester, H. Schulzrinne, V. Hilt, S. Seetharaman, and A. Khan, "NetServ: dynamically deploying in-network services," in *ReArch '09: Proceedings of the 2009 workshop on Re-architecting the internet*. New York, NY, USA: ACM, 2009, pp. 37–42.

[9] Netfilter queue. http://www.netfilter.org/projects/libnetfilter_queue/index.html.

[10] S. A. Baset, J. Reich, J. Janak, P. Kasparek, V. Misra, D. Rubenstein, and H. Schulzrinne, "How Green is IP-Telephony?" in *The ACM SIGCOMM Workshop on Green Networking*, 2010.

[11] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner, "Router plugins: a software architecture for next-generation routers," *IEEE/ACM Trans. Netw.*, vol. 8, no. 1, pp. 2–15, 2000.

[12] S. Schmid, J. Finney, A. Scott, and W. Shepherd, "Component-based active network architecture," in *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on*, 2001, pp. 114 –121.

[13] R. Keller, L. Ruf, A. Guindehi, and B. Plattner, "Promethos: A dynamically extensible router architecture supporting explicit routing," in *IWAN '02: Proceedings of the IFIP-TC6 4th International Working Conference on Active Networks*. London, UK: Springer-Verlag, 2002, pp. 20–31.

[14] G. Hjalmtysson, "The pronto platform: a flexible toolkit for programming networks using a commodity operating system," in *Open Architectures and Network Programming, 2000. Proceeding. OPENARCH 2000. 2000 IEEE Third Conference on*, Mar. 2000, pp. 98 –107.