2003

VIOLIN: Virtual Internetworking on Overlay Infrastructure

Xuxian Jiang

Dongyan Xu
Purdue University, dxu@cs.purdue.edu

Report Number:
03-027

VIOLIN: VIRTUAL INTERNETWORKING ON OVERLAY INFRASTRUCTURE

Xuxian Jiang
Dongyan Xu

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #03-027
July 2003

VIOLIN: Virtual Internetworking on Overlay Infrastructure

Xuxian Jiang, Dongyan Xu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
{jiangx, dxu}@cs.purdue.edu

Abstract

We propose a novel alternative to application-level overlays called VIOLIN, or Virtual Internetworking on OverLay INfrastructure. Inspired by recent advances in virtual machines, VIOLINs are virtual and isolated networks created on top of an overlay infrastructure such as PlanetLab. Entities in a VIOLIN include virtual routers, switches, and end-hosts, all implemented in software and hosted by physical overlay hosts. The salient features of VIOLIN include: (1) Each VIOLIN is a 'virtual world' with its own IP address space. And its activities and communications are strictly confined within the VIOLIN. (2) All VIOLIN entities can be created, deleted, or migrated on-demand. (3) It provides a new playground to deploy, leverage, and evaluate value-added network services which are not widely deployed in the real Internet. An application can simply connect to a VIOLIN and leverage the network services provided. (4) It releases application developers from network service implementation details, resulting in easier application implementation and maintenance. We have designed and implemented a prototype of VIOLIN in PlanetLab.

1 Introduction

To maintain efficiency and scalability, the Internet only provides the most basic network services such as IP unicast. In recent years, overlay networks have emerged as an application-level solution to the realization of value-added network services, such as anycast, multicast, reliable multicast, and active networking. While highly practical and effective, application-level overlays pose a number of issues: (1) The implementation of application functions and network services are often closely coupled, making the development and management of overlays complicated with *blurred* boundary between application and network functions. (2) The development of network services in application-level overlays is mainly individual efforts, leading to few standard and reusable protocols. Meanwhile, many network services for transport, routing, and management [4, 11, 12, 14, 22], which have been well designed in the past decade, are left *under-leveraged*. (3) At application level, it is hard to achieve strong *isolation* between an overlay and the rest of the Internet. For example, a compromised overlay node can potentially attack any host in the Internet.

In this paper, we propose a novel alternative to application-level overlays called VIOLIN, or Virtual Internetworking on OverLay INfrastructure. VIOLIN is inspired by recent advances in virtual machines [7, 23] and has been deployed in the PlanetLab [16] overlay infrastructure. The idea is to create virtual and isolated internetworking environments, each called a VIOLIN[1], on top of an overlay infrastructure. A VIOLIN is a small-scale virtual network with virtual routers, LANs, and end-hosts, all implemented in software and hosted by overlay hosts. Network protocols for routing, transport, and management can run unmodified in a VIOLIN as in the real Internet. The key difference between VIOLINs and application-level overlay is that VIOLIN re-introduces *system(OS)-enforced* boundary between applications and network services. As a result, distributed applications running in a VIOLIN will be simple to develop and manage, while value-added network protocols will have a chance to be deployed and used - at their intended (network) level.

The salient features of VIOLIN include: (1) Each VIOLIN is a 'virtual world' with its own IP address space. And most activities and communications are strictly confined within the VIOLIN[2]. (2) Every entity in a VIOLIN is implemented in software. In this sense, VIOLIN has more deployment flexibility than a real network by allowing on-demand addition/deletion/migration/configuration of routers, switches and end-hosts. (3) From the perspective of networking, VIOLIN provides a new playground to deploy, leverage, and evaluate both existing and emerging network services which may not be available in the real

[1] With a slight abuse of terms, VIOLIN stands for both the virtual internetworking technique and a virtual internetwork based on the technique.

[2] We use the word 'most' because we also implement a regulated and optional *jenway* to connect the virtual world and real Internet.

Internet. An application can simply connect to a VIOLIN and leverage the network services provided. (4) From the perspective of distributed applications, it releases application developers from network service implementation details, resulting in easier application implementation and maintenance.

We expect VIOLIN to be a useful complement to application-level overlays. First, VIOLIN can be used to create testbeds for experiments with network-level algorithms and protocols. Such a testbed comprises more realistic network entities and topology, and provides researchers with more convenience in experiment setup and dynamic re-configuration. Second, VIOLIN can be used to create a service-oriented (virtual) IP network with advanced network services such as IP multicast and anycast. Virtual end-hosts running distributed applications will then join the VIOLIN and enjoy these network services which are not widely available in the real Internet. It can be imagined that the operator of an overlay infrastructure may set up a regional or nation-wide service-oriented VIOLIN to serve 'customers', i.e. distributed applications such as video conferencing, on-line community, and peer selection.

We have designed and implemented a prototype of VIOLIN in PlanetLab. We have also developed a number of sample applications to demonstrate the benefit of VIOLIN. The rest of the paper is organized as follows. Section 2 provides an overview of VIOLIN. Section 3 justifies the design of VIOLIN and its benefit to distributed applications. Section 4 describes the implementation and ongoing research problems of VIOLIN. Section 5 compares VIOLIN with related works. Finally, section 6 concludes this paper.

2 VIOLIN Overview

The concept of VIOLIN is illustrated in Figure 1. The low-level plane is a real network; the mid-level plane is an overlay infrastructure such as PlanetLab; and the top-level plane is one VIOLIN created on the overlay infrastructure. All entities in the VIOLIN are hosted by overlay hosts; and there are three types of entities like in the real network: end-host, LAN, and router.

- A *virtual end-host (vHost)* is a virtual machine in a physical overlay host. Meanwhile, it is possible that one physical overlay host supports multiple vHosts belonging to different VIOLINs.

- A *virtual LAN (vLAN)* is constructed by creating one *virtual switch* (*vSwitch*, not shown in Figure 1) that connects multiple vHosts. *Different* from a real LAN, vHosts on a vLAN may be geographically dispersed. It is even possible to create a vLAN where vHosts have organizational or social proximity rather than network proximity.

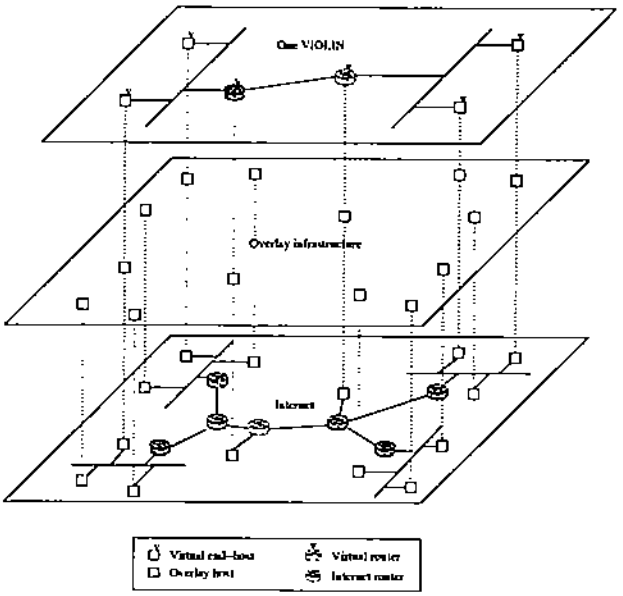

Figure 1. An overview of VIOLIN and its relation to Internet and overlay infrastructure

- A *virtual router (vRouter)* is also a virtual machine with multiple *virtual NICs (vNICs)*. A vRouter interconnects several vLANs and performs packet forwarding among them.

Based on existing and our own virtual machine and networking techniques, these entities form a small-scale virtual network with its own IP address space[3]. Network protocols run in the (virtual) OS of the VIOLIN entities. Figure 2 shows a simple VIOLIN we create in PlanetLab. Two vLANs are interconnected by one vRouter (vRouter1 hosted by *planetlab1.cs.purdue.edu*): One vLAN comprises vHost1, vHost2, and vSwitch1; while the other one comprises vHost3, vHost4, and vSwitch2. The links between these entities emulate cables in the real world. Furthermore, with the all-software implementation of VIOLIN, the migration and re-wiring of vSwitches, vRouters, and vHosts can be performed easily.

3 VIOLIN Design Justification

In this section, we make the case for VIOLIN and describe how applications (including network experiments) can benefit from VIOLIN.

[3]Such address space can even *overlap* that of the real Internet, if no VIOLIN-Internet communications are needed.

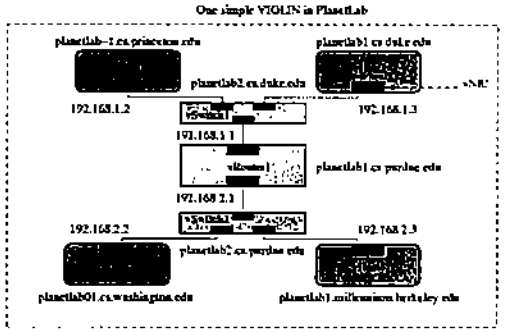

Figure 2. A VIOLIN in PlanetLab (with names of physical PlanetLab hosts and virtual IP addresses)

3.1 Virtualization and Isolation

Analogous with the relation between virtual machine and its host machine, VIOLIN achieves strong virtualization of the real network, as well as strong isolation between a VIOLIN and the rest of the Internet.

Virtualization makes it possible to run unmodified Internet protocols in VIOLINs: if the implementation of a protocol works in the real network, it will also work in a VIOLIN. Furthermore, entities in a VIOLIN are totally *unaware* of the underlying Internet. For example, if we perform *traceroute* from vHost1 (hosted by planetlab-1.cs.princeton.edu) to vHost3 (hosted by planetlab01.cs.washington.edu) in Figure 2, we will only see vRouter1 as the intermediate router and the hop count is two, although the PlanetLab hosts at Princeton and at UW are many more hops apart in the real Internet. More interestingly, it is potentially feasible to repeat such virtualization *recursively*: a level-n VIOLIN can be created on a level-$(n-1)$ VIOLIN, with level-0 being the real Internet[4].

Isolation is with respect to (1) administration: the creator of a VIOLIN has full administrator privilege - but *only* within this VIOLIN; (2) address space and protocol: the IP address spaces of two VIOLINs can safely overlap and the versions and implementations of their network protocols can be different - for example, one running IPv4 while the other running IPv6; (3) attack and fault impact: due to (1) and (2), any attack or fault in one VIOLIN will not affect the rest of the Internet; (4) resources: *if* the underlying overlay infrastructure provides QoS support [19, 20], VIOLIN will be able to achieve resource isolation for local resources (such as CPU and memory [10]) of VIOLIN entities and for network bandwidth between them.

Benefit to applications System-level virtualization and isolation provide a confined and dedicated environment for untrusted distributed applications and risky network experiments (such as router-aware intrusion detection and wide-area live virus monitoring). From another perspective, applications requiring strong confidentiality can use VIOLIN to prevent both internal information leakage and external attacks.

[4] However, to implement recursive VIOLINs, we need a non-trivial and well-crafted memory address space layout among applications and different layers of virtual machines.

3.2 System-Enforced Layering

Contrary to the all-layers-in-one design of application-level overlays, VIOLIN features strong layering enforced by the (virtual) OS of VIOLIN entities. Layering is a common technique to disentangle the design and implementation of application and network functions, making VIOLIN more maintainable and extensible at both levels. In addition, OS-enforced layering provides better protections to network services after the application level is compromised.

We note that layering itself does *not* incur additional performance overhead, compared with application-level overlays. It is virtualization, the technique to *enforce* layering that introduces the main overhead of VIOLIN, similar to the case of virtual machines versus real machines. We also note that layering is between application and network functions, *not* between network protocols. In fact, VIOLIN can be used as a testbed for the *protocol heap* architecture [3].

Benefit to applications Application developers will be able to focus on application functions rather than implementation details of network services, leading to clean design and easy implementation. In addition, applications that work in the Internet will also work in a VIOLIN.

3.3 Network Service Provisioning

VIOLIN provides a new opportunity to deploy and evaluate network services other than the basic ones in the current Internet. There exist a large number of well-designed network protocols which for some reason have not been widely adopted. Examples include IP multicast, scalable reliable multicast [11, 14], IP anycast [12], and active networking [4, 22]. There also exist protocols that are still in the initial stage of incremental deployment such as IPv6. VIOLIN is a convenient platform to make these protocols a (virtual) reality.

Benefit to applications VIOLIN allows applications to take full advantage of value-added network services. For example, in a VIOLIN capable of IP multicast, applications such as publish-subscribe, layered media broadcast can be more easily developed and installed than in the real Internet. We further envision the emergence of *service-oriented* VIOLINs, each with high-performance vRouters and vSwitches deployed at strategic locations (for example, vRouters close to Internet routing centers, vSwitches close

to domain gateways), so that customers can connect their applications to such a VIOLIN (instead of to the Internet) for advanced network services. Operators of such VIOLINs will compete with each other by introducing newer and better network services.

3.4 Easy Reconfigurability

Based on all-software virtualization techniques, VIOLIN achieves easy reconfigurability. Different from a real network, vRouters, vSwitches and vHosts can be added, removed, or migrated dynamically. Also, vNICs can be dynamically added to or removed from vHosts or vRouters; and the number of ports supported by a vSwitch is no longer a hardware constraint. Instead, it is constrained by the capacity of the underlying overlay host.

Benefit to applications The easy reconfigurability and hot vNIC plug-and-play capability of VIOLIN is especially useful to handle the dynamic load and/or membership of distributed applications. Not only can a VIOLIN be created/torn down on-demand for an application, its scale and topology can also be adjusted in a demand-driven fashion. For example, during a multicast session, a new vLAN can be dynamically grafted on a vRouter to accommodate more participants.

4 VIOLIN Implementation

This section presents the key building blocks of VIOLIN implementation: *virtual machine*, *virtual switch*, and *virtual router*. Current status and ongoing work of VIOLIN will also be discussed.

4.1 Virtual Machine

All VIOLIN entities are implemented as virtual machines in overlay hosts. To achieve universal deployment, the virtual machine (VM) technology should impose minimum requirement on the underlying *host OS* of overlay hosts[5]. Especially, since we implement VIOLIN in PlanetLab, *no* host OS kernel modification is allowed.

We adopt User-Mode Linux (UML) [6] as the VM technology. UML allows most Linux-based applications to run on top of it without any modification[6]. Based on *ptrace* mechanism, UML - the *guest OS* for a virtual machine, performs system call redirection and signal handling to emulate a real OS. More specifically, the guest OS will be notified when an application running in the virtual machine issues a system call, the guest OS will then redirect the system call into its own implementation and nullify the original call. One important feature of UML is that it is completely implemented at user level without requiring host OS kernel modifications[7]. Finally, with careful memory layout of UML kernels and applications, UML is able to support *recursive* virtual machines, and thus recursive VIOLINs.

Unfortunately, the original UML has a serious limitation: both virtual NICs and virtual links of virtual machines are restricted *within* the same physical host. *Inter-host* virtual links, which are essential to VIOLIN, have not been reported in current VM projects [1, 7, 23]. To break the physical host boundary, we have performed non-trivial extension to UML and introduced transport-based *inter-host tunneling*. More specifically, we use UDP tunneling in the Internet domain to emulate the physical layer in the VIOLIN domain. For example, to emulate the physical link between a vHost and a vSwitch, the guest OS for the vHost opens a UDP transport connection for the vNIC and obtains a file descriptor - both in the host OS domain. To receive data from the vSwitch, SIGIO signal will be generated by the host OS for the file descriptor whenever data are available. The vSwitch maintains the IP address and UDP port number (in the Internet domain) for the vNIC of the vHost, so that the vSwitch can correctly emulate data link layer frame forwarding. Such virtualization is transparent to the network protocol stack in the guest OS. Finally, inter-host tunneling enables hot plug-and-play of vNICs (Section 3.4); and it does not exhibit MTU effect as in the EtherIP [8] and IP-in-IP [21] approaches.

In addition to inter-host tunneling, *intra-host tunneling*, also dubbed the *jetway* connecting VIOLIN and Internet, is also provided as a VIOLIN *option* that can be dynamically turned on/off (it should be off when strong VIOLIN isolation is required). Intra-host tunneling creates a TUN/TAP pseudo interface in an overlay host, which serves as the access point of a VIOLIN to the real Internet and vice versa. In addition to a TUN/TAP pseudo interface, a *virtual bridge* may also be created so that the pseudo interface becomes visible from the Internet, which eliminates the need for the overlay host to act as a router or proxy.

4.2 Virtual Switch

A vSwitch is created for each vLAN and is responsible for packet forwarding at the data link layer (in the VIOLIN domain). Figure 3 shows a vSwitch which connects multiple vHosts. vSwitch is emulated by a UDP daemon in the host OS domain. The *poll* system call is used to poll the arrival of data and perform data queuing, forwarding, or dropping. More delicate link characteristics may also be implemented in the UDP daemon. The *poll* system call also notifies the UDP daemon of the arrival of a connect request

[5] For this reason, we do not adopt VMWare[1] because it requires VMWare installation in every overlay host.

[6] Exceptions are applications that involve privileged instructions, such as *hwclock* using *iopl* and *inb/outb*.

[7] However, certain modifications, such as *skas* mode [6], can further improve the performance of UML.

from a new vHost joining the vLAN, so that a new port can be created for the vHost, as shown in Figure 3. Due to the all-software implementation, there is no hardware constraint on the number of ports created; and it is possible to migrate a vSwitch.

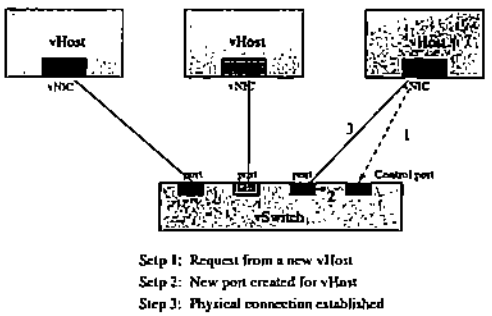

Figure 3. vSwitch and steps of port creation

4.3 Virtual Router

Interestingly, there is no intrinsic difference in implementation between vHost and vRouter, except that the latter has additional packet forwarding capability and user level routines for the configuration of packet processing policies. Linux source tree makes it possible to accommodate versatile and extensible packet processing capabilities.

When a UML is bootstrapped, a recognizable file system will be located and mounted as root file system. Based on UML, the vRouter requires kernel-level support for the capability of packet forwarding, as well as user-level routines, namely *route, iproute2, ifconfig* for the configuration of interface addresses and routing table entries. Beyond the packet forwarding capability, it is also easy to add firewall, NAT, and other value-added services to the UML kernel.

In the VIOLIN implementation, we adopt the *zebra* [9] open-source routing package, which provides a relatively complete suite of routing protocol implementations. We observe in our experiments that vRouters, with jetway and virtual bridge enabled, can even exchange routing table information with real Internet routers using OSPF protocol. Recently, to enable active network services, we have also incorporated Click [13] as an optional package for vRouters.

4.4 Current Status and Ongoing Work

We have implemented a basic prototype of VIOLIN, as well as a couple of sample applications that run in VIOLIN. The prototype has been deployed in PlanetLab. More specifically, our VIOLIN prototype provides IP multicast service. We have developed (1) a streaming video multicast application[8] and (2) a publish-subscribe application, both with multiple vHosts distributed in the wide-area PlanetLab, to demonstrate the convenience and effectiveness of leveraging the network service (IP multicast) in VIOLIN. We are continuing to refine and improve VIOLIN, as well as to perform extensive evaluations and measurements.

Our ongoing work includes:

- *Performance evaluation and comparison* VIOLIN involves virtualization techniques and is based on the overlay infrastructure. How to evaluate the performance, resilience, and adaptability of VIOLIN, compared with the real Internet and with application-level overlays? Especially, to match the performance of application-level overlays, how much additional computation and communication capacity need to be allocated? Our video multicast application in VIOLIN demonstrates performance comparable to its counterpart in an application-level overlay. However, more in-depth evaluation and measurement are needed before these questions can be answered.

- *Refinement of network virtualization technique* Our inter-host tunneling implementation is initial and there is plenty of room for refinement and improvement. For example, how to improve the reliability of virtual links? Should we adopt another transport protocol (such as TCP), or integrate error correction (such as FEC) into UDP, or simply let the transport protocols in the VIOLIN domain to achieve reliability? To monitor the status of virtual links, is it possible to leverage the *routing underlay* [15] for better Internet friendliness?

- *Topology planning and optimization* Our implementation provides mechanisms for dynamic VIOLIN topology setup and adjustment. However, we have not studied the the problem of VIOLIN topology planning and optimization. More specifically, given the overlay infrastructure, where to place the vRouters and vSwitches, in order to achieve Internet bandwidth efficiency and satisfactory application performance? How should a VIOLIN react to the dynamics of Internet condition and application workload using its dynamic reconfigurability (Section 3.4)?

5 Related Work

VIOLIN is made possible by PlanetLab [16], an open and global overlay infrastructure for the deployment and assessment of planetary-scale network services. PlanetLab itself provides resource virtualization capability called slicing. Compared with slicing, VIOLIN virtualization has a more specific goal: to create an isolated internetworking environment where entities are *unaware* of the underlying Internet. Netbed [24] is another testbed for experiments

[8] Currently, default PlanetLab hosts do not support remote X Windows access. We re-direct video display to non-PlanetLab machines.

with networks and distributed systems, and VIOLIN can potentially be deployed in Netbed too.

Application-level overlays have been proved highly feasible and effective in realizing value-added network services. For example, RON [2] achieves robust routing and packet forwarding for application end-hosts; and the Narada protocol [5] brings high network efficiency to end system multicast. As discussed earlier, VIOLIN is proposed as an alternative and complement to application-level overlays, especially to those requiring strong network virtualization and isolation. In fact, there are cases where application-level solutions *are* more effective, such as the lookup service [17, 18] in *structured* P2P networks.

Machine virtualization has recently received tremendous attention. VMWare [1] fully virtualizes the PC hardware, while Denali [23] and Xen [7] take the *paravirtualization* approach by creating a virtual machine similar (instead of identical) to the physical machine. Inspired by machine virtualization, VIOLIN is our initial effort toward *network* virtualization.

The X-Bone [21] provides automated deployment and remote monitoring of overlays, and allows network entities (hosts, routers) to participate in multiple overlays simultaneously. Their approach of two-layer IP-in-IP tunneled overlay exhibits smaller MTU effect. It also makes physical IP (in the real Internet domain) visible to entities in the overlay domain, resulting in lower degree of isolation and virtualization than VIOLIN.

6 Conclusion

We present VIOLIN as a novel alternative and useful complement to application-level overlays. Based on all-software virtualization techniques, VIOLIN creates a virtual internetworking environment for the deployment of advanced network services, with no modification to the real Internet infrastructure. The properties of isolation, enforced-layering, and easy reconfigurability make VIOLIN an excellent platform for the development and execution of network experiments and distributed applications, which is demonstrated by our implementation of VIOLIN prototype and sample applications in PlanetLab.

References

[1] VMWare. *http://www.vmware.com*.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. *Proc. of ACM SOSP 2001*, Oct. 2001.

[3] R. Braden, T. Faber, and M. Handley. From Protocol Stack to Protocol Heap Role-Based Architecture. *Proc. of ACM HotNets-I*, Oct. 2002.

[4] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz. Directions in Active Networks. *IEEE Communications Magazine*, Oct. 1998.

[5] Y. H. Chu, S. G. Rao, and H. Zhang. A Case For End System Multicast. *Proc. of ACM SIGMETRICS 2000*, June 2000.

[6] J. Dike. User Mode Linux. *http://user-mode-linux.sourceforge.net*.

[7] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. *Proc. of ACM SOSP 2003*, Oct. 2003.

[8] R. Housley and S. Hollenbeck. EtherIP: Tunneling Ethernet Frames in IP Datagrams. *http://www.faqs.org/rfcs/rfc3378.html*, Sept. 2002.

[9] K. Ishiguro. Zebra. *http://www.zebra.org/*.

[10] X. Jiang and D. Xu. vBET: a VM-Based Emulation Testbed. *Proc. of ACM SIGCOMM 2003 Workshops (MoMeTools)*, Aug. 2003.

[11] S. Kasera, G. Hjalmtysson, D. Towsley, and J. Kurose. Scalable Reliable Multicast Using Multiple Multicast Channels. *IEEE/ACM Trans. on Networking*, June 2000.

[12] D. Katabi and J. Wroclawski. A Framework for Scalable Global IP-Anycast (GIA). *Proc. of ACM SIGCOMM 2000*, Aug. 2000.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. on Computer Systems*, Aug. 2000.

[14] C. Liu, D. Estrin, S. Shenker, and L. Zhang. Local Error Recovery in SRM: Comparison of Two Approaches. *IEEE/ACM Trans. on Networking*, Dec. 1998.

[15] A. Nakao, L. Peterson, and A. Bavier. Routing Underlay for Overlay Networks. *Proc. of ACM SIGCOMM 2003*, Aug. 2003.

[16] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *Proc. of ACM HotNets-I*, Oct. 2002.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. *Proc. of ACM SIGCOMM 2001*, Aug. 2001.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *Proc. of ACM SIGCOMM 2001*, Aug. 2001.

[19] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: a Scalable Architecture to Approximate Fair Bandwidth Allocations in High-speed Networks. *IEEE/ACM Trans. on Networking*, 11(1), 2003.

[20] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: Offering Internet QoS Using Overlays. *Proc. of ACM HotNets-I*, Oct. 2002.

[21] J. Touch. Dynamic Internet Overlay Deployment and Management Using the X-Bone. *Proc. of IEEE ICNP 2000*, Nov. 2000.

[22] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: Network Services without the Red Tape. *IEEE Computer*, 32(4), 1999.

[23] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. *Proc. of USENIX OSDI 2002*, Dec. 2002.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *Proc. of USENIX OSDI 2002*, Dec. 2002.