VIOLIN: Virtual Internetworking on Overlay Infrastructure

Xuxian Jiang and Dongyan Xu

Purdue University, West Lafayette, IN 47907, USA
{jiangx, dxu}@cs.purdue.edu

Abstract. We propose a novel application-level virtual network architecture called VIOLIN (Virtual Internetworking on OverLay INfrastructure). VIOLINs are isolated virtual networks created on top of an overlay infrastructure (e.g., PlanetLab). Entities in a VIOLIN include virtual end-hosts, routers, and switches implemented by software and hosted by physical overlay hosts. Novel features of VIOLIN include: (1) a VIOLIN is a "virtual world" with its own IP address space. All its computation and communications are strictly confined within the VIOLIN. (2) VIOLIN entities can be created, deleted, or migrated on-demand. (3) Value-added network services not widely deployed in the real Internet can be provided in a VIOLIN. We have designed and implemented a prototype of VIOLIN in PlanetLab.

1 Introduction

Current Internet only provides basic network services such as IP unicast. In recent years, overlay networks have emerged as application-level realization of value-added network services, such as anycast, multicast, reliable multicast, and active networking. While highly practical and effective, overlays have the following problems: (1) Application functions and network services are often closely coupled in an overlay, making the development and management of overlays complicated. (2) The development of overlay network services is mainly individual efforts, leading to few standards and reusable protocols. Meanwhile, advanced network services [1][2][3][4][5] have been developed but not widely deployed. (3) It is hard to *isolate* an overlay from the rest of the Internet, making it easy for a compromised overlay node to attack other Internet hosts.

In this paper, we propose a novel virtual network architecture called VIOLIN (Virtual Internetworking on OverLay INfrastructure), motivated by recent advances in virtual machine technologies [6][7]. The idea is to create virtual isolated network environments on top of an overlay infrastructure. A VIOLIN[1] consists of virtual routers, LANs, and end-hosts, all being software entities hosted by overlay hosts. The key difference between VIOLIN and application-level overlay is that VIOLIN re-introduces *system(OS)-enforced* boundary between applications

[1] With a slight abuse of terms, VIOLIN stands for either the virtual network technique or one such virtual network.

and network services. As a result, a VIOLIN becomes an "advanced Internet" running value-added network-level protocols for routing, transport, and management.

The novel features of VIOLIN include: (1) Each VIOLIN is a "virtual world" with its own IP address space. All its computation and communications are strictly confined within the VIOLIN. (2) All VIOLIN entities are software-based, leading to high flexibility by allowing on-demand addition/deletion/migration/configuration. (3) Value-added network services not widely deployed in the real Internet can be provided in a VIOLIN. (4) Legacy applications can run in a VIOLIN without modification, while new applications can leverage the advanced network services provided in VIOLIN.

We expect VIOLIN to be a useful complement to application-level overlays. First, VIOLIN can be used to create testbeds for network-level experiments. Such a testbed contains more realistic network entities and topology, and provides researchers with more convenience in experiment setup and configuration. Second, VIOLIN can be used to create a service-oriented (virtual) IP network with advanced network services such as IP multicast and anycast, which will benefit distributed applications such as video conferencing, on-line community, and peer selection.

We have designed and implemented a prototype of VIOLIN in PlanetLab. A number of distributed applications have also been deployed in 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 presents preliminary performance measurements in PlanetLab. Section 6 compares VIOLIN with related works. Finally, section 7 concludes this paper and outlines our ongoing work.

2 VIOLIN Overview

The concept of VIOLIN is illustrated in Figure 1. The low-level plane is the real IP network; the mid-level plane is an overlay infrastructure such as PlanetLab; and the top-level plane shows 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 running 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.
- A *virtual router (vRouter)* is also a virtual machine with multiple *virtual NICs (vNICs)*. A vRouter interconnects two or more vLANs.

Figure 2 shows a simple VIOLIN we create in PlanetLab. Two vLANs are interconnected by one vRouter (vRouter1 hosted by *planetlab1.cs.purdue.edu*):

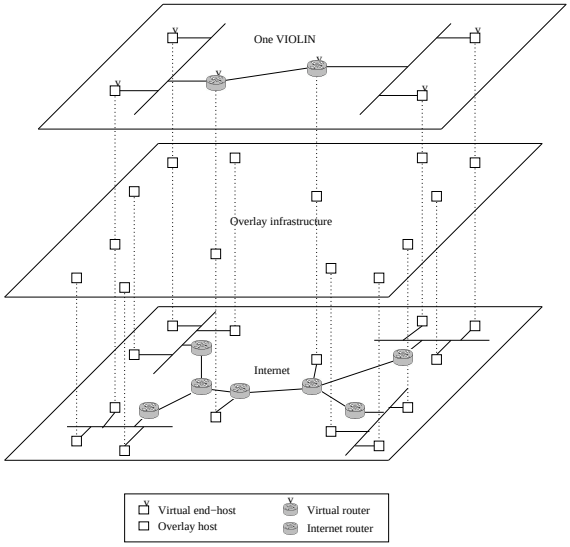


Fig. 1. VIOLIN, overlay infrastructure, and underlying IP network

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. The IP address space of the VIOLIN is completely independent. Therefore, it can safely overlap the address space of another VIOLIN or the real Internet.

3 VIOLIN Design Justification

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

3.1 Virtualization and Isolation

Analogous with the relation between virtual machine and its host machine, VIOLIN involves network virtualization and leads to isolation between a VIOLIN and the underlying IP network. Virtualization makes it possible to run unmodified Internet protocols in VIOLINs. Furthermore, entities in a VIOLIN are decoupled from 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 actual 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.

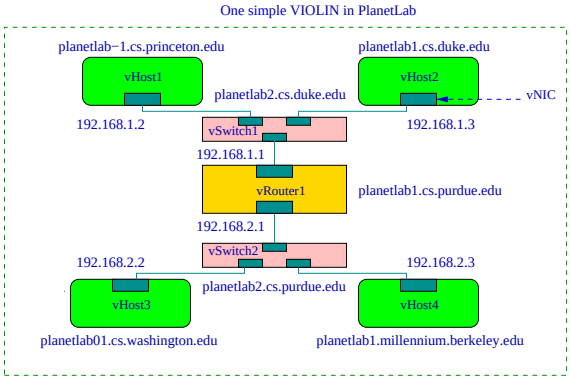


Fig. 2. A VIOLIN in PlanetLab (with names of physical PlanetLab hosts and virtual IP addresses)

Network isolation is with respect to (1) administration: the owner 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: 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 [8][9], 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. From another perspective, applications requiring strong confidentiality can use VIOLIN to prevent both internal information leakage and external attacks.

3.2 System-Enforced Layering

Contrary to application-level overlays, VIOLIN enforces strong layering in order to disentangle application functions and network services. In addition, OS-enforced layering provides better protections to network services after the application level software is compromised. We note that layering itself does *not* incur more performance overhead compared with application-level overlays. 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 [11].

Benefit to applications Application developers will be able to focus on application functions rather than network services, leading to clean design and easy

implementation. In addition, legacy applications can run in a VIOLIN without modification and re-compilation.

3.3 Network Service Provisioning

VIOLIN provides a new opportunity to deploy and evaluate advanced network services. There exist a large number of well-designed network protocols that are not yet widely deployed. Examples include IP multicast, scalable reliable multicast [2][4], IP anycast [3], and active networking [1][5]. There are also protocols that are still in the initial stage of incremental deployment (e.g., IPv6). VIOLIN is a 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 conveniently developed 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 clients can connect to the VIOLIN to access its advanced network services.

3.4 Easy Reconfigurability

Based on all-software virtualization techniques, VIOLIN achieves easy reconfigurability. Different from a physical 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.

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

4.1 Virtual Machine

All VIOLIN entities are implemented as virtual machines (VMs) in overlay hosts. We adopt User-Mode Linux (UML) [12] as the VM technology. UML allows most Linux-based applications to run on top of it without any modification. 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.

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 [6][7][13]. 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 [14] and IP-in-IP [15] approaches.

4.2 Virtual Switch

A vSwitch is created for each vLAN and is responsible for packet forwarding at the (virtual) data link layer. 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 from a new vHost joining the vLAN, so that a new port can be created for the vHost, as shown in Figure 3.

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

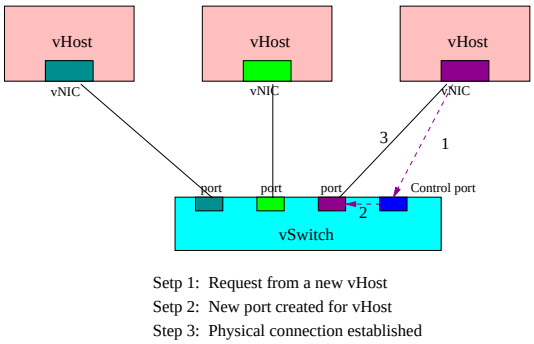


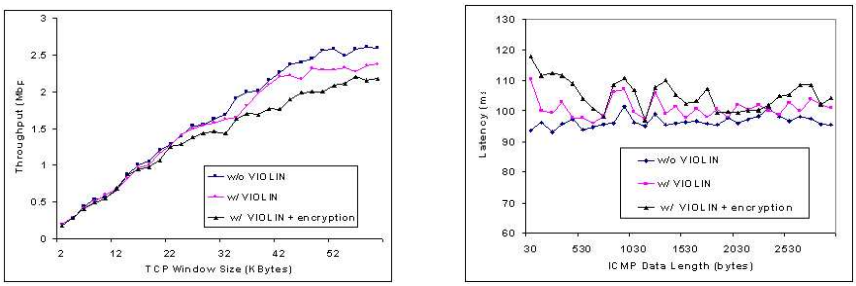
Fig. 3. vSwitch and steps of port creation

VIOLIN implementation, we adopt the *zebra* [16] open-source routing package, which provides a comprehensive suite of routing protocol implementations. Recently, to enable active network services, we have also incorporated Click [17] as an optional package for vRouters.

5 VIOLIN Performance

We have implemented a VIOLIN prototype and deployed it in PlanetLab. To evaluate the performance of VM communications in a VIOLIN, we have performed end-to-end throughput and latency measurement between VMs. Figures 4(a) and 4(b) show a set of representative results. Two VMs are hosted by PlanetLab nodes planetlab8.lcs.mit.edu and planetlab6.cs.berkeley.edu, respectively. We measure the TCP throughput and ICMP latency between the VMs, with and without the vSwitches performing UDP payload encryption. As a comparison, we also measured the TCP throughput and ICMP latency between the two PlanetLab hosts. Our measurement results show that VIOLIN introduces an average of 5% degradation in TCP throughput, compared with the TCP throughput between the two underlying physical hosts. The addition degradation due to VM traffic encryption is 5% on the average. The degree of ICMP latency degradation (increase) is even less than that of TCP throughput.

To demonstrate VIOLIN's support for advanced network services, we run WaveVideo, a legacy video streaming application, in VIOLIN. WaveVideo requires IP multicast and therefore is not runnable in PlanetLab. However, WaveVideo is able to execute in a VIOLIN with 9 VMs. The VM hosted by planetlab2.cs.wisc.edu is the source of the video multicast session. It streams a short 300-frame video clip using (virtual) IP multicast address 224.0.0.5. The other 8 VMs are all receivers in three different domains: Princeton, Purdue, and Duke. The average peak signal noise ratios (PSNR) of video frames observed by VMs in the three domains are shown in Figure 5.



(a) TCP throughput (b) ICMP latency

Fig. 4. TCP throughput and ICMP latency between two VMs in VIOLIN

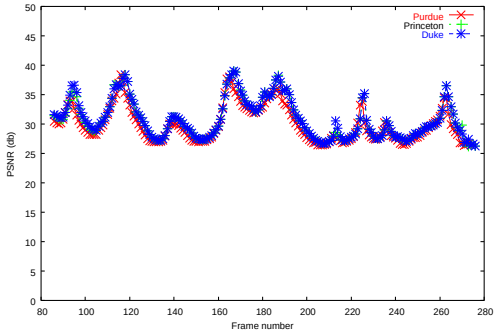


Fig. 5. Video streaming quality in an IP-multicast-enabled VIOLIN

6 Related Work

VIOLIN is made possible by PlanetLab [18], which itself provides resource virtualization capability called slicing. Netbed [19] is another wide-area testbed for network and distributed system experiments. Because of its high portability, VIOLIN can also be deployed in Netbed.

Application-level overlays have achieved significant success in recent years. For example, RON [20] achieves robust routing and packet forwarding for application end-hosts; and the Narada protocol [21] brings high network efficiency to end system multicast. VIOLIN is proposed as an alternative and complement to application-level overlays, especially for legacy applications or untrusted applications that require strong network confinement.

Machine virtualization has recently received tremendous attention. VMware [13] fully virtualizes the PC hardware, while Denali [7] and Xen [6] 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 [15] provides automated deployment and remote monitoring of overlays, and allows network entities (hosts, routers) to participate in multiple overlays simultaneously. By taking the two-layer "IP-in-IP" tunneling approach, X-Bone makes real Internet IPs visible to entities in the overlay domain, leading to a lower degree of isolation and confinement than VIOLIN.

7 Conclusion and Ongoing Work

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 modifications to the Internet infrastructure. The properties of isolation, enforced-layering, and easy reconfigurability make VIOLIN an excellent platform for the execution of high-risk network experiments, legacy applications unaware of overlay APIs, as well as untrusted and potentially malicious applications. 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* [22] 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)?

Acknowledgments

We thank the anonymous reviewers for their reviews. This work is supported in part by the National Science Foundation (NSF) under the grant SCI-0438246.

References

1. Calvert, K., Bhattacharjee, S., Zegura, E., Sterbenz, J.: Directions in Active Networks. IEEE Communications Magazine (1998)
2. Kasera, S., Hjalmtysson, G., Towsley, D., Kurose, J.: Scalable Reliable Multicast Using Multiple Multicast Channels. IEEE/ACM Trans. on Networking (2000)
3. Katabi, D., Wroclawski, J.: A Framework for Scalable Global IP-Anycast (GIA). Proc. of ACM SIGCOMM 2000 (2000)
4. Liu, C., Estrin, D., Shenker, S., Zhang, L.: Local Error Recovery in SRM: Comparison of Two Approaches. IEEE/ACM Trans. on Networking (1998)
5. Wetherall, D., Guttag, J., Tennenhouse, D.: ANTS: Network Services without the Red Tape. IEEE Computer **32** (1999)
6. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R.: Xen and the Art of Virtualization. Proc. of ACM SOSP 2003 (2003)
7. Whitaker, A., Shaw, M., Gribble, S.D.: Scale and Performance in the Denali Isolation Kernel. Proc. of USENIX OSDI 2002 (2002)
8. Stoica, I., Shenker, S., Zhang, H.: Core-Stateless Fair Queueing: a Scalable Architecture to Approximate Fair Bandwidth Allocations in High-speed Networks. IEEE/ACM Trans. on Networking **11** (2003)
9. Subramanian, L., Stoica, I., Balakrishnan, H., Katz, R.: OverQoS: Offering Internet QoS Using Overlays. Proc. of ACM HotNets-I (2002)
10. Jiang, X., Xu, D.: vBET: a VM-Based Emulation Testbed. Proc. of ACM SIGCOMM 2003 Workshops (MoMeTools) (2003)
11. Braden, R., Faber, T., Handley, M.: From Protocol Stack to Protocol Heap Role-Based Architecture. Proc. of ACM HotNets-I (2002)
12. Dike, J.: User Mode Linux. (http://user-mode-linux.sourceforge.net)
13. : VMware. (http://www.vmware.com)
14. Housley, R., Hollenbeck, S.: EtherIP: Tunneling Ethernet Frames in IP Datagrams. http://www.faqs.org/rfcs/rfc3378.html (2002)
15. Touch, J.: Dynamic Internet Overlay Deployment and Management Using the X-Bone. Proc. of IEEE ICNP 2000 (2000)
16. Ishiguro, K.: Zebra. (http://www.zebra.org/)
17. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The Click Modular Router. ACM Trans. on Computer Systems (2000)
18. Peterson, L., Anderson, T., Culler, D., Roscoe, T.: A Blueprint for Introducing Disruptive Technology into the Internet. Proc. of ACM HotNets-I (2002)
19. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An Integrated Experimental Environment for Distributed Systems and Networks. Proc. of USENIX OSDI 2002 (2002)
20. Andersen, D.G., Balakrishnan, H., Kaashoek, M.F., Morris, R.: Resilient Overlay Networks. Proc. of ACM SOSP 2001 (2001)
21. Chu, Y.H., Rao, S.G., Zhang, H.: A Case For End System Multicast. Proc. of ACM SIGMETRICS 2000 (2000)
22. Nakao, A., Peterson, L., Bavier, A.: Routing Underlay for Overlay Networks. Proc. of ACM SIGCOMM 2003 (2003)