Port-Space Isolation for Multiplexing a Single IP Address through Open vSwitch*

Ping Du[1], Maoke Chen[1], and Akihiro Nakao[2]

[1] National Institute of Information and Communications Technology (NICT), Japan
[2] The University of Tokyo, Japan

Abstract. Large-scale network testbeds raise the problem of the exhaustion of IPv4 address space. Before the IPv6 is widely deployed, multiplexing IPv4 address for guest slivers is necessary. NAT is one of the typical ways for the multiplexing. Violating the end-to-end feature of the Internet, the NAT approach has well-known drawbacks in performance scalability and in supporting diverse services and applications. In this paper, we propose a method to share the host's global IP address for all the guest slivers on a node and isolate their network usage in port-space. The idea is successfully implemented with Open vSwitch and deployed in the CoreLab platform. Benchmark result shows that the proposed solution is superior to NAT technique significantly.

Keywords: Network virtualization, resource isolation, Open vSwitch, testbed infrastructure.

1 Introduction

Large-scale network testbeds, such as PlanetLab [1], OneLab [2], EmuLab [3], CoreLab [4,5], etc., are widely deployed over the world, enabling researchers to run their experiments simultaneously without affecting each other's. Different requirements for resource isolation lead to different designs for testbed infrastructures. PlanetLab employs a resource container called Linux-VServer [6] for offering an isolated execution environment, while EmuLab loads arbitrary operating systems on bare hardware on demand. Our CoreLab applies both kernel-based virtual machine (KVM) [7] as a hosted virtual machine monitor (VMM) and OpenVZ [8] as a resource container.

No matter what kind of virtualization is applied for the resource isolation, testbeds are bound to face a common problem: it is necessary to isolate each sliver [1] from the others. In today's Internet, IP address is playing the role of identifying a logical peer over the Internet, and therefore a testbed node needs

* This work has been partly supported by Ministry of Internal Affairs and Communications (MIC) of the Japanese Government.

[1] Throughout this paper, we use the PlanetLab jargon such as *slice* and *sliver*. A slice is a collection of slivers distributed across the Internet, and a sliver is a unit of resources for enabling an execution environment.

T. Magedanz et al. (Eds.): TridentCom 2010, LNICST 46, pp. 113–122, 2011.

a large number of IP addresses to accommodate hundreds of slivers. IPv6 is an apparent solution; however, the current distribution of IPv6 is still sparse so that using IPv6 is still a nuisance. Therefore, multiplexing a global IPv4 address becomes a necessity in the deployment of a testbed.

Assigning private addresses to guest slivers and applying network address translation (NAT) is one of the typical solution for the IPv4 address multiplexing. It is well-known that the NAT violates the end-to-end context, although the commercial use of NAT is already prevalent. For network research testbeds, however, the violation of end-to-end brings obvious limitations. First, when a new application layer protocol designed by the researchers carries an IP address, the NAT fails to translate the address in application messages unless a new patch for NAT is applied, making it an application-layer gateway. Second, when both peers are located behind NAT boxes and using non-well-known port numbers for their services, they cannot communicate with each other unless they inform the NAT boxes to set up port mapping for them. Third, when there are problems during experiments, the additional indirection and address translations increases the time and coordination needed to debug and solve those problems. Fourth, keeping states of address-port mapping also makes NAT solution not scalable with heavy overhead.

In this paper, we propose an alternative way of IPv4 address multiplexing among slivers without NAT, attempting to directly applying the host's address for all the guest slivers and isolating their network activities through static separation of the port-space. We explore both the workaround with the `ebtables` [9] and the full-fledged solution with programming the Open vSwitch (or, briefly, OVS) [10]. We verify our solution in the practice of a real platform — the CoreLab, and compare the performance scalability with the previous NAT solution. The idea of multiplexing global IP address without translation is also used in PlanetLab but there the port-space is not statically isolated, which is not suitable for the case other than containers that share the system calls of the host.

The rest of the paper is organized as follows. Section 2 summarizes the work of NAT-based multiplexing and PlanetLab's dynamic sharing. Section 3 discusses our design challenges and proposes the workaround based on the `ebtables` tool and the OVS-based solution. Section 4 presents the experimental results of our proposal. Finally, Section 5 concludes our work.

2 Related Work

2.1 PlanetLab

PlanetLab applies Linux VServer [6], where each guest sharing the host kernel and all network resources such as CPU, memory are maintained in the same way as in ordinary OSes. Currently, all slivers on a PlanetLab node use the same IP address [11]. Each time, when a sliver is launching a session, the host selects an unused TCP/UDP port as its source port. Since all guests share the same kernel, it is easy for kernel to detect and avoid port conflict.

When there is port contention that more than two slivers want to bind the same ports simultaneously, a solution could be scheduling or using a resource allocator like SHARP [12] to arbitrate the port usage.

The shortcoming for above solutions is that current PlanetLab can only support short-term usage of a port. This is not sufficient if one would like to hold a port for a relatively long period.

Applying Linux VServer for PlanetLab brings another limitation in flexibility. Sharing the system calls of the host, slivers cannot run with a customized kernel, while adding new kernel modules often happens in the research for networking.

2.2 CoreLab

CoreLab applies KVM to support the slivers, getting rid of the limitations in flexibility. In the previous practice of CoreLab [5], each guest sliver is assigned a private IPv4 address and a specific range of port numbers for either TCP or UDP communications. Meanwhile the host OS is configured as a NAT gateway, translating the public address to proper guest's private address according to the destination port in any packet. The structure of deployment is depicted in Fig. 1. Therefore, the configuration includes a series of `iptables` DNAT (destination NAT) rules. For example, on a host with IP address 133.69.37.10, a guest OS has the private IP address 10.0.6.2 and runs an `sshd` service on the port 22, while port range 10000 ˜ 10100 is assigned to this guest. The host maps the address-port pair 10.0.6.2:22 to 133.69.37.10:10022, and a remote peer can access the guest's `sshd` service through the global address 133.69.37.10 and the port number 10022.

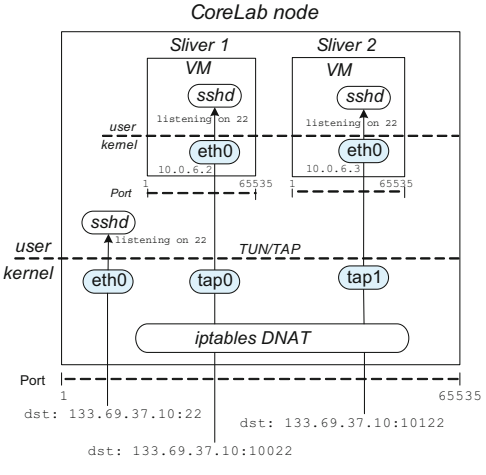

Fig. 1. Sharing IP address through iptables with NAT on CoreLab node

3 System Design

As we have mentioned before, sharing IP address through NAT has two shortcomings: (1) NAT breaks the end-to-end connectivity so that many applications

are unavailable; (2) NAT is a bottleneck of network performance since each session requires a piece of state recorded in the translator.

Current Internet distinguishes end systems (either physical or virtual) with IP address. One observation is that, though the 32-bit address defined by IPv4 is insufficient, the 16-bit port-space is inefficiently utilized. Based on this observation, we propose a solution that distinguishes slivers through port range instead of IP address, where different guest slivers can have the same address while using different ranges of ports.

3.1 Problem and Challenges

Because we don't incline to use the NAT mode to manage the connectivity between co-located virtual machines and the physical interface, the host should be configured in the bridge mode. Sharing the same IP address among the host itself and all its guests is equivalent to a case of IP anycast in the same connected network. If we had nothing done, once a packet that contains the anycast IP address as the destination comes, who (guest or host) would respond to the packet depends upon who would earliest receive the packet and process it.

We attempt to deliver a packet to a proper guest according to the port range that the packet's destination port falls in. This problem is equivalent to optimizing the anycast response according to port number. Today a typical IP anycast solution is announcing the anycast address via routing protocols and who's selected depends on the distance in routing. Such a solution is obviously not suitable for our case. We need an anycast optimization solution for a group of computers bridged together. Because the criteria for the optimization involve the transportation layer — the port number, our goal is similar to design a Layer-4 switch with anycast optimization. Xen [13] and Vmware [14] provides standard Layer-2 switching or Layer-3 routing functionality for a virtual environment. However, they are also lack of the Layer-4 switching functionality. This is a generic, unsolved problem.

In details, we face the following challenges.

1. For the last hop towards one in the anycast group, the link-layer address learned by the link layer peer decides the real destination. Unfortunately, however, ARP is only a protocol mapping Layer-3 address to Layer-2 but not a mapping considering port number. It is also a "slow" protocol whose cache entries have quite long lifetime, not able to be frequently updated — once a peer have learned the MAC address of one of the guests (or the MAC address of the host) for the shared IP address, it has to take some time to update to the MAC address of another.
2. There are some IP-supported protocols, like ICMP, that do not have "port" number in the payload of the IP datagram. Semantically, it is reasonable that one in the anycast group can receive all the non-port messages that are actively sent from outside and the non-port responses that are replied to its own requests.
3. Most TCP and UDP applications are designed so that a client randomly chooses source port number for a session that is not locally conflicting with

other existing sessions. Such a port number may not fall into the range of ports assigned to the system.
4. Some applications use a separated channel for data transmission rather than signaling. The port number for the data channel is often negotiated at the application layer and it is hard to limit an application program selecting only assigned ports for the negotiation without changing the program binary.

In order to resolve these challenges, we design two solutions, a bridge-based solution and a dynamically programmable switch. Although they are tested on the KVM-based CoreLab, we believe they are also suitable for any virtualization approach with independent kernel for guests, such as Xen and VMware running in a bridge mode.

3.2 *Ebtables* and Workaround

A straightforward idea is to redirect packets towards a guest according to the port number. The redirection is similar to the behavior of *iptables* but it should be done in the Layer-2. Therefore, the Ethernet filter *ebtables* [9] is considered.

Having the same IP address, each guest (as well as the host itself) can be distinguished from others through the MAC address. Therefore it is needed to establish a mapping from port range to a MAC address. The *ebtables* provides MAC address translation that fits the requirement. People may concern the scalability of the translation. Fortunately, the MAC address translation is fairly simple and each guest system involves only a static set of states, which is much lighter than IP NAT that stores per-session states.

Deploying the *ebtables* for the port-space isolation contains a couple of steps.

First, we hack the incoming frame with our rule in order a packet is able to be received by the proper guest. This is done with the destination MAC address translation of the *ebtables*.

Second, to avoid updating ARP entries for a peer, we disable any guest to respond ARP requests from the peer, defining the host as the only agent for itself and all its guests. For this purpose, we change the outgoing frame's source MAC address, and also let only the host itself reply ARP request.

ARP frame has the source hardware address twice, in both the frame header and also the ARP message body. It is necessary to change not only the guest MAC address in frame header but also that in the message body for any ARP frame.

However, the second action also disables a guest to receive ARP replies from an outside peer. Once an ARP reply from a peer arrives, we cannot recognize if this is requested by any guest or by the host itself. What we have to do is only delivering the reply to all of them by changing the ARP reply's destination address to the Ethernet broadcast address.

The above configurations haven't overcome the ICMP challenge. A workaround is forcing ICMP response to be broadcasted to all the guests, but the privacy is hurt. Furthermore, they haven't overcome the challenge of random port selection or that of the application-layer port negotiation. If application

chooses a port not available, now the system simply rejects the communication. A possible workaround is occupying the unassigned ports with a special daemon, similar to the `inetd`. Thus the application will automatically choose an available port. However, this method brings extra overhead.

3.3 Open vSwitch Solution

Open vSwitch (OVS) [10] can work as a network switch for virtual environment that hosts inside a physical machine and connects the various VMs. It provides a flexible, a 10-tuple flow-table [15] based forwarding engine which can to used to partition the forwarding plane. OpenFlow [16] can also provide a similar flow-table forwarding model. We applied OVS since it is more compatible with Linux-based virtual environment. With OVS, we can define flow entries to forward packets based on its port number.

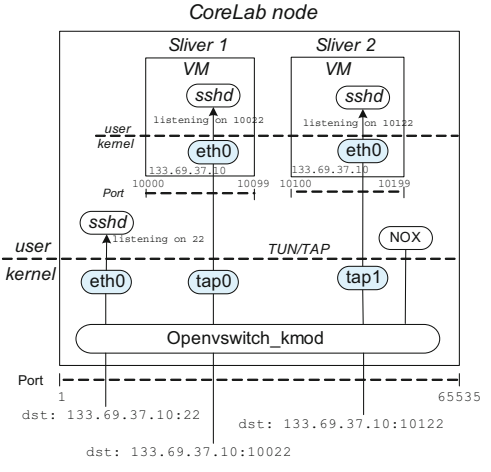

Fig. 2. Port-space isolation with OVS on CoreLab node

We design and deploy OVS solution in the practice of a real platform — the CoreLab. As shown in Fig. 2, all VMs and host are bridged to a datapath (a kind of bridge) of OVS. Besides OVS, we also deploy deployed a NOX [18], which is an open-source controller that speaks OpenFlow protocol on each CoreLab node. The flow entries are installed from NOX to OVS. We address the challenges as follows.

a. Port-space isolation

Since the virtual interface of each VM is written in software, all VMs can be configured with the same IP and MAC addresses as the host so that any ethernet frames from outside can be received by a VM or host without address translation. This can reduce the overhead of NAT and etables that require maintaining state of address translation. For the ARP packets, since all VMs share the same IP and MAC addresses with the host, when an ARP packet is received by an host, it will be flooded to all VMs.

To isolate the packets of different VMs, each VM is assigned with a range of port numbers. The port range of each VM can be got from the database of PLC [17] node. As a result, the corresponding flow entries (forwarding rules) are installed after a VM is launched. Each VM can only listen on the ports that assigned to it. The OVS switches packet based on the destination port number. For example, an ssh request packet with <dst: 133.69.37.10:10022> is delivered to VM 1 while the one with <dst: 133.69.37.10:10122> is delivered to VM 2.

b. ICMP packets

ICMP packets that do not have "port" number in the payload of IP datagram. When a host receives an ICMP request packet, it will respond it without redirecting it to VMs. When a VM sends out an ICMP request packet, it will install a flow entry for the reverse incoming ICMP response packets. For example, when VM 1 sends ICMP packets to remote host B, it will install a flow entry <nw_src=B,icmp_echo_response, actions: VM1>. The flow entry will expire after it has been idle for a specified period (i.e., 10 secs). When VM 2 is sending ICMP packets to a different remote host C, the ICMP response packets could be separated.

One possible problem is that when two VMs are sending ICMP packets to the same host B, the flow entry will be <nw_src=B,icmp_echo_response, actions: VM1|VM2>. In this case, all ICMP packets will be sent to both VM1 and VM2. An ideal solution is to install flow entry based on icmp_id. However, due to that current OpenFlow protocol does not support user-defined flow-space, we can redefine the unused vlan_id field as icmp_id. This is left for our future work.

c. Source port conflict

We can reduce the port conflict probability to zero by changing the VM configuration file (i.e., /proc/sys/net/ipv4/ip_local_port_range for Linux) to let the TCP/UDP can only select source port in a specified range. We also apply a simple patch to the VM kernel to prevent the VM from selecting a source port out of the range.

d. Multi-homing

When a host has multiple physical interfaces (and therefor multiple IP addresses), we create the same number of OVS datapaths. Each datapath is bridged to a physical interface. A guest VM boots with multiple virtual interfaces bridged to different datapaths. As a result, each guest VMs can share one or multiple global IP addresses with host. How to deliver packets through these multiple virtual interfaces is decided by each guest separately, which is out of the scope of this paper.

4 Performance Evaluation

In this section, we compare the performance of our implemented port-space isolation with OVS against the previous practice with NAT on CoreLab.

Figure 3 shows the experimental environment, which is configured with two CoreLab nodes. Each of them is with a 2.67GHz Intel CPU and 4GB memory.

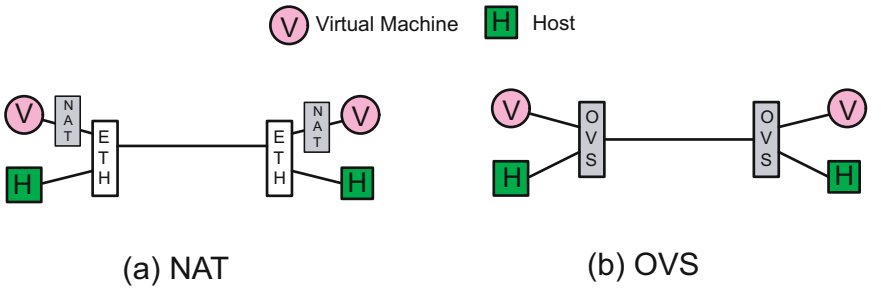

Fig. 3. Experimental environment for evaluating port-space isolation with (a) NAT (b) OVS

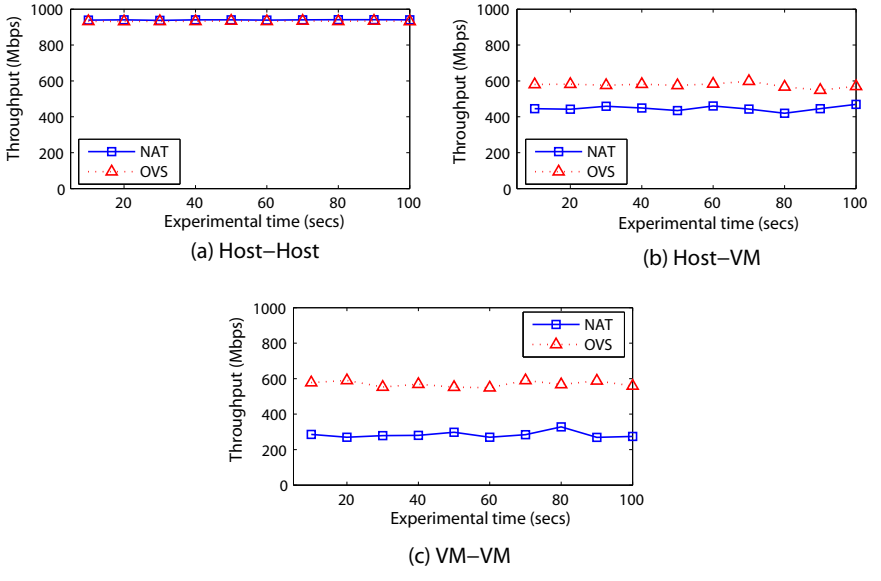

Fig. 4. Comparisons of throughput of (a) Host-Host (b) Host-VM, and (c) VM-VM under NAT and OVS solutions

They connect with each other over 1Gbps Ethernet link. The host OS is Fedora 8 with kernel 2.6.31. Each VM is with 512M memory and its virtual interface driver is `ne1000`. In the NAT case (Fig. 3(a)), the VM connects outside through NAT while the host connects outside directly. In OVS case (Fig. 3(b)), Both VM and host connect outside through OVS implementation in the Linux kernel.

We run `iperf` server and client on VM or host of different nodes and measure the effective TCP throughput between them. Figure 4(a) compares the throughput between two hosts. In the case of NAT, since the hosts are connected directly so that the throughput are that of two nodes of native Linux. In the case of OVS, the hosts are connected through OVS implementation in the Linux kernel. Both of them can achieve around 940Mbps throughput. The results show that OVS has very small performance cost comparable to that of native OS.

Figure 4(b) compares the throughput between a host and a VM. In the case of NAT, a packet of a VM will go through both NAT and KVM. As a comparison, in the case of OVS, a packet of a VM will go through NAT and OVS. Under the same cost of KVM, the throughput is around 650Mbps in the case of OVS while the throughput is only 400Mbps in the case of NAT. The results show that OVS has much smaller performance cost than NAT.

Figure 4(c) compares the throughput between two VMs at different Corelab nodes. In the case of NAT, the throughput (250Mbps) between two VMs is much smaller than the throughput (450Mbps) of Fig. 4(b) since the packets should go through NAT gateway one more time. As a comparison, in the case of OVS, the throughput (600Mbps) between two VMs is almost the same as the throughput between a host and a VM.

In the case of OVS, although the performance of cost of OVS is very small, the throughput (600Mbps) between two VMs is much lower than that (940Mbps) between two hosts. This is due to the performance cost of KVM, which requires improvements.

5 Conclusions

This study has identified the IP scarcity problem in a testbed environment. Since conventional NAT solution has many limitations. We have sought to solve the problem by port-space isolation. We have designed and implemented such a port-space isolation system through Open vSwitch on CoreLab. The experiment results show that it has better performance than conventional NAT technique.

In fact, the IP scarcity problem exists not only in a testbed environment, but also in most stub network such in an office LAN. The techniques of private IP address or IPv6 can make the Internet become heterogeneous. To make the Internet remain flat, this paper proposes a new angle for identifying and routing packets by ports instead of IP addresses in a local environment.

As a final note, we posit that this work can be immediately extended to generic network-namespace isolation for slices. In other words, while this paper shows how to isolate a port range for a slice, the other network-namespace such as IP addresses, protocols, and the other fields as well as a tuple of them can be allocated to a slice. OpenFlow [16] and Open vSwitch [10] allow us to partition network by flows and to define simple actions per flow at the central controller [18]. In CoreLab, we will combine network partitioning—not only in terms of only port range, but also of a generic flow space—and computational slicing together to enabling complex processing in a distributed fashion. Such extension is our immediate future work.

References

1. Planetlab, `http://www.planet-lab.org/`
2. Onelab, `http://www.onelab.eu/`

3. 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. In: OSDI (2002)
4. Corelab, http://www.corelab.jp
5. Nakao, A., Ozaki, R., Nishida, Y.: Corelab: An emerging network testbed employing hosted virtual machine monitor. In: ROADS (2008)
6. Linux-vserver project, http://linux-vserver.org/
7. Kvm, http://kvm.qumranet.com/kvmwiki
8. Openvz, http://wiki.openvz.org/
9. Ebtables, http://ebtables.sourceforge.net/
10. Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., Shenker, S.: Extending networking into the virtualization layer. In: ACM HotNets (2009)
11. Port use and contention in planetlab, https://www.planet-lab.org/files/pdn/PDN-03-016/pdn-03-016.pdf
12. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: Sharp: An architecture for secure resource peering. In: ACM SOSP (2003)
13. Dragovic, P.B.B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP (2003)
14. Vmware vnetwork distributed switch, http://www.vmware.com/products/vnetwork-distributed-switch/
15. Openflow switch specification, http://www.openflowswitch.org/documents/openflow-spec-v0.8.9.pdf
16. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J.: Openflow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38, 69–74 (2008)
17. The trutees of princeton university. myplc, http://www.planet-lab.org/doc/myplc
18. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: Nox: towards an operating system for networks. SIGCOMM Computer Communication Review 38, 105–110 (2008)