NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters

Jayaram Mudigonda
Praveen Yalagandula
Jeff Mogul
HP Labs, Palo Alto, CA

Bryan Stiekes
Yanick Pouffary
HP

ABSTRACT

Providers of "Infrastructure-as-a-Service" need datacenter networks that support multi-tenancy, scale, and ease of operation, at low cost. Most existing network architectures cannot meet all of these needs simultaneously.

In this paper we present NetLord, a novel multi-tenant network architecture. NetLord provides tenants with simple and flexible network abstractions, by fully and efficiently virtualizing the address space at both L2 and L3. NetLord can exploit inexpensive commodity equipment to scale the network to several thousands of tenants and millions of virtual machines. NetLord requires only a small amount of offline, one-time configuration. We implemented NetLord on a testbed, and demonstrated its scalability, while achieving order-of-magnitude goodput improvements over previous approaches.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation, Management

Keywords

Datacenter Network, Network Virtualization, Multi-Tenant, Multi-Pathing, Scalable Ethernet

1. INTRODUCTION

Cloud datacenters such as Amazon EC2 [1] and Microsoft Azure [6] are becoming increasingly popular, as they offer computing resources at a very low cost, on an attractive *pay-as-you-go* model. Many small and medium businesses are turning to these cloud computing services, not only for occasional large computational tasks, but also for their IT jobs. This helps them eliminate the expensive, and often very complex, task of building and maintaining their own infrastructure.

Cloud datacenter operators, on the other hand, can provide cost-effective Infrastructure as a Service (IaaS), because they can time-multiplex the physical infrastructure among a large number of *tenants*. The advent of mature CPU virtualization techniques (e.g., VMWare [26] and Xen [28]) makes it possible to convert the dedicated, and often extremely underutilized, physical servers in an enterprise into *Virtual Machines* (VMs) that run in an IaaS datacenter.

To fully realize the benefits of resource sharing, these datacenters must scale to huge sizes. The larger the number of tenants, and the larger the number of VMs, the better the chances for multiplexing, which in turn achieves better resource efficiency and cost savings.

Increasing the scale alone, however, cannot fully minimize the total cost. Today, a great deal of expensive human effort is required to configure the equipment, to operate it optimally, and to provide ongoing management and maintenance. A good fraction of these human costs reflect the complexity of managing a multi-tenant network; IaaS datacenters cannot become cost-effective at scale unless we can reduce these costs.

Therefore, IaaS networks must support virtualization and multi-tenancy, at scales of tens of thousands of tenants and servers, and hundreds of thousands of VMs. They must keep costs down by exploiting commodity components and by facilitating automatic configuration and operation. Most existing datacenter network architectures, however, suffer one or more of the following drawbacks:

They are expensive to scale: Today, scaling the network to the sizes needed by IaaS datacenters remains very expensive. The straightforward scaling of existing datacenter networks requires huge core switches with thousands of ports [8]. Some approaches require complex new protocols to be implemented in hardware [5, 17], or may work only with specific features such as IP-in-IP decapsulation [13] and MAC-in-MAC encapsulation [5, 17]. Some approaches that do not require switch modifications (e.g., [18]) may require excessive switch resources – in particular, they require very large forwarding tables, because the MAC address of every VM is exposed to the switches. None of these architectures can easily leverage existing, inexpensive commodity switches.

They provide limited support for multi-tenancy: Ideally, a multi-tenant network should provide a network abstraction that allows a tenant to design its network as if it were the sole occupant of a datacenter. That is, a tenant should be able to define its own layer-2 (L2) and layer-3 (L3) addresses. Previous multi-tenancy architectures do not provide full address-space virtualization; they either focus on performance guarantees and performance isolation [14, 24, 25], or only provide IP address space sharing [11, 13].

They require complex configuration: Many existing architectures that might make use of cheaper switches often require careful manual configuration: for example, setting up IP subnets and configuring OSPF [8, 13].

In this paper, we present *NetLord*, a novel multi-tenant virtualized datacenter network architecture. NetLord encapsulates a tenant's L2 packets, to provide full address-space virtualization. NetLord employs a light-weight agent in the end-host hypervisors to transparently encapsulate and decapsulate packets from and to the local VMs. Encapsulated packets are transferred over an underlying, multi-path L2 network, using an unusual combination of IP and Ethernet packet headers. NetLord leverages SPAIN's approach to multi-pathing [18], using VLAN tags to identify paths through the network.

The encapsulating Ethernet header directs the packet to the destination server's edge switch, via L2 forwarding. By leveraging a novel configuration of the edge switch's IP forwarding table, the encapsulating IP header then allows the switch to deliver the packet to the correct server, and also allows the hypervisor on that server to deliver the packet to the correct tenant.

Because NetLord does not expose any tenant MAC addresses to the switches (and also hides most of the physical server addresses), the switches can use very small forwarding tables, thus reducing capital costs, while scaling to networks with hundreds of thousands of VMs. Because NetLord uses simple, static switch configurations, this reduces operational costs.

Our focus in this paper is only on *qualitative* isolation between tenants: tenants can design their L2 and L3 address spaces without any restrictions created by multi-tenancy. NetLord itself provides no guarantees on *performance* isolation between tenants [14, 24, 25]. However, because NetLord explicitly exposes tenant identifiers in the encapsulation header, it can efficiently support various slicing/QoS mechanisms using commodity switches.

In this paper, we present the NetLord design, and show through simple calculations how it can scale. We also describe an experimental evaluation, with up to 3000 tenants and 222K emulated VMs, showing that NetLord can achieve substantial goodput improvements over other approaches.

2. PROBLEM AND BACKGROUND

We start by describing the problems we are trying to solve, and then describe some prior work on similar problems.

2.1 Goals: scalable, cheap, and flexible

Our goal is to provide a network architecture for a multi-tenant virtualized cloud datacenter. We want to achieve large scale at low cost, with easy operation and configuration, and we want to provide tenants with as much flexibility as possible.

Scale at low cost: Cloud providers can offer services at low cost because they can leverage economies of scale. This implies that the network for a cloud datacenter must scale, at low cost, to large numbers of tenants, hosts, and VMs. The network must support lots of addresses, and provide ample bandwidth between the VMs of any tenant.

A provider's costs include both capital expenditures (CAPEX) and operational expenditures (OPEX). To reduce CAPEX, the network should use commodity, inexpensive components. However, commodity network switches often have only limited resources and limited features. For example, typical commercial switches can hold a few tens of thousands of MAC forwarding information base (FIB) table entries in their data-plane fast paths (e.g., 64K entries in the HP ProCurve 6600 series [20] and 55K in the Cisco Catalyst 4500 series [10]).

Small data-plane FIB tables in switches create a scaling problem for MAC-address learning: if the working set of active MAC addresses is larger than the data-plane table, some entries will be lost, and a subsequent packet to those destinations will cause flooding. (Even when the table is large enough, a rapid arrival rate of new addresses can lead to flooding, if learning is done in software and the switch's local management CPU is too slow to keep up.) Therefore, we cannot afford to expose the switches to the MAC addresses of all tenant VMs, or even of all physical servers in a large network, because the resultant flooding will severely degrade performance. Section 5.3 demonstrates this effect experimentally.

We also want to be able to support high bisection bandwidth at low cost. In particular, we would like to allow the cloud provider to choose an efficient and cost-effective physical wiring topology without having to consider whether this choice interferes with multi-tenancy mechanisms or tenant-visible abstractions.

Easy to configure and operate: To reduce OPEX, cloud providers need networks that, as much as possible, can be configured and operated automatically. We would like to avoid any per-switch configuration that is hard to scale, or that is highly dynamic. We also want to avoid network-related restrictions on the placement of VMs, to allow the cloud provider to efficiently multiplex physical hosts, without worrying about resource fragmentation.

Cloud providers may wish to offer QoS features to tenants. We would also like to provide simple mechanisms that support per-tenant traffic engineering; we do not want to require the provider to individually manage QoS for each TCP flow for each tenant.

We also want the network to handle switch and link failures automatically. We would like the unfailed portions of the network to continue to work without being affected by failed components.

Flexible network abstraction: Different tenants will have different network needs. A tenant wishing to run a Map-Reduce job might simply need a set of VMs that can communicate via TCP. On the other hand, a tenant running a three-tier Web application might need three different IP subnets, to provide isolation between tiers. Or a tenant might want to move VMs or entire applications from its own datacenter to the cloud, without needing to change the network addresses of the VMs. This flexibility will also allow tenants to create networks that span VMs both in their own datacenters or on rented servers in the cloud datacenter [15].

These examples, and others, motivate our desire for a datacenter network that fully virtualizes the L2 and L3 address spaces for each tenant, without any restrictions on the tenant's choice of L2 or L3 addresses.

Also, in certain kinds of cloud environments, tenants might wish to use non-IP protocols, such as Fibre Channel over Ethernet (FCoE), ATA over Ethernet (AoE), or HyperSCSI. These protocols, while currently unsupported in public cloud networks, could be important for tenants trying to move existing applications into the cloud, and would be impossible to use in a network that did not support an L2 abstraction. Similarly, these tenants would benefit from a cloud network that supports tenant-level broadcasts or muliticasts.

2.2 State of the art

In this section, we first describe current practices and recent research proposals for multi-tenant datacenter networking. Also, since NetLord depends on an underlying large-scale L2 network, we discuss recent work on scalable network fabrics.

Multi-tenant datacenters: Traditionally, datacenters have employed VLANs [16] to isolate the machines of different tenants on a single L2 network. This could be extended to virtualized datacenters, by having the hypervisor encapsulate a VM's packets with a

VLAN tag corresponding to the VM's owner. This simple approach provides an L2 abstraction to the tenants, and can fully virtualize the L2 and L3 address spaces. However, to correctly support the Spanning Tree Protocol, each VLAN needs to be a loop-free subgraph of the underlying network, and that limits the bisection bandwidth for any given tenant. Also, unless VLANs are carefully laid out, this approach may expose all VM addresses to the switches, creating scalability problems. Finally, the VLAN tag is a 12-bit field in the VLAN header, limiting this to at most 4K tenants. (The IEEE 802.1ad standard on Provider Bridges [3] defines the "QinQ" protocol to allow stacking of VLAN tags, which would relieve the 4K limit, but QinQ is not yet widely supported.)

Amazon's Virtual Private Cloud (VPC) [2] provides an L3 abstraction and full IP address space virtualization. Tenants can specify up to 20 arbitrary IP subnets (at least /28 in size), and the provider will instantiate a VPC router to connect these IP subnets. We could not find any documentation of how the VPC router is implemented, and hence cannot comment on its routing efficiency. VPC does not support multicast and broadcast, which implies that the tenants do not get an L2 abstraction.

Greenberg *et al.* [13] propose VL2, a scalable and flexible datacenter network. VL2 provides each tenant (termed "service" in the paper) with a single IP subnet (L3) abstraction, but implements efficient routing between two different IP subnets without needing to divert the packets through an explicit router. Services running in VL2 are expected to use only IP-based protocols. VL2 works with a specific topology (Clos) to achieve a full bisection bandwidth network, and hence does not restrict VM placement. However, the approach assumes several features not common in commodity switches, such as IP-in-IP decapsulation support at line rates. VL2 handles a service's L2 broadcast packets by transmitting them on a IP multicast address assigned to that service. The VL2 paper does not explicitly address unicast non-IP packets, but we believe their approach can be extended, as it encapsulates all packets.

Diverter [11] presents an efficient fully-distributed virtualized routing system, which accommodates multiple tenants' logical IP subnets on a single physical topology. Similar to VL2, Diverter addresses the problem of efficient routing between subnets. Diverter's solution is to overwrite the MAC addresses of inter-subnet packets, allowing it to relay these packets via a single hop. Diverter provides an L3 network abstraction to tenants, but it assigns unique IP addresses to the VMs; that is, it does not provide L3 address virtualization.

Scalable network fabrics: Many research projects and industry standards address the limitations of Ethernet's Spanning Tree Protocol. Several, including TRILL (an IETF standard) [5], Shortest Path Bridging (an IEEE standard) [4], and Seattle [17], support multipathing using a single-shortest-path approach. These three need new control-plane protocol implementations and data-plane silicon. Hence, inexpensive commodity switches will not support them, for at least a few years.

One way to achieve scalable multipathing is through hierarchical addressing in specific topologies. Al-Fares *et al.*[7] proposed three-level FatTree topologies, combined with a specific IP address assignment scheme, to provide high bisection bandwidth without needing expensive, high-radix core switches. For scalability, their proposal depends on a two-level route lookup feature in the switches. Mysore *et al.* proposed PortLand [19], which replaces that IP address scheme with MAC-address rewriting, and requires switches with the ability to forward based on MAC-address prefixes. Both these approaches work only with multi-rooted tree topologies.

Scott *et al.* proposed MOOSE [22], which address Ethernet scaling issues by using hierarchical MAC addressing. MOOSE also uses shortest-path routing, and did not focus on multipathing for improved bisection bandwidth. MOOSE, like PortLand, needs switches that can forward packets based on MAC prefixes.

Mudigonda *et al.* [18] proposed SPAIN, which uses the VLAN support in existing commodity Ethernet switches to provide multipathing over arbitrary topologies. SPAIN uses VLAN tags to identify *k* edge-disjoint paths between pairs of endpoint hosts. The original SPAIN design may expose each end-point MAC address *k* times (once per VLAN), stressing data-plane tables even more than standard Ethernet, and hence it can not scale to large number of VMs.

To summarize, no current practices or prior research proposals meets all of the goals we described in section 2.1.

3. NETLORD'S DESIGN

The fundamental idea underlying NetLord is to encapsulate the tenant's L2 packets and transfer them over a scalable L2 fabric, using an L3+L2 (IP+Ethernet) encapsulation that exploits features of both layers. NetLord uses a light-weight agent in the hypervisors to encapsulate, route, decapsulate, and deliver tenant packets to virtual machines, addressed to the VMs' tenant-assigned Ethernet addresses. With two exceptions, described in sections 3.2 and 3.9, NetLord ignores any tenant-visible L3 (IP) issues.

The source NetLord Agent (NLA) creates the encapsulating L2 and L3 headers such that the Ethernet destination address directs the packet through the underlying L2 fabric to the correct edge switch, and such that the IP destination address both allows the egress edge switch to deliver the packet to the correct server, and allows the destination NLA to deliver the packet to the correct tenant. The details of this encapsulation are somewhat subtle; we discuss them in section 3.5.

One significant consequence of this encapsulation method is that tenant VM addresses are never exposed to the actual hardware switches. By using IP forwarding on (only) the last hop, we can effectively share a single edge-switch MAC address across a large number of physical and virtual machines. This resolves the problem of FIB-table pressure; in NetLord the switches, instead of needing to store millions of VM addresses in their FIBs, only need to store the addresses of the other switches. Even in a very large datacenter, we expect at most a few thousand switches. (Edge-switch FIBs must also store the addresses of their directly-connected server NICs – at most, one per switch port; they do *not* need to store the addresses of remote servers.)

Because of the specific encapsulation used in NetLord, edge switches require very limited configuration; in particular, this configuration requires only local information, plus some boilerplate configuration that is essentially identical on every switch. This aspect of NetLord dramatically simplifies the operation and configuration of the network hardware, obviates the need for complex routing protocols, and reduces the chances for software or human error to create failures. We describe the details of configuration in section 3.6.

Another consequence of NetLord's encapsulation method is that it exposes tenant IDs in the outer L3 header of packets moving through the fabric. This potentially allows a cloud provider to do per-tenant traffic management in the network fabric, without having to put per-flow ACLs in the switches, which would create significant scaling problems. We discuss tenant-level network management in section 6.

Since we intend NetLord for use in a large-scale datacenter, the underlying L2 fabric needs multi-path support, for high band-

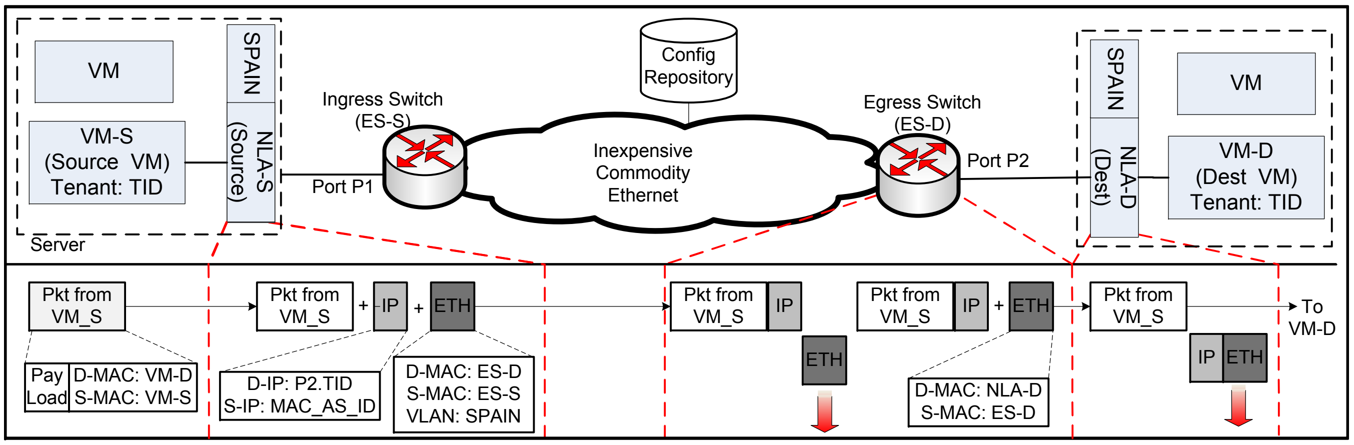


Figure 2: NetLord's high-level component architecture (top) and packet encapsulation/decapsulation flows (bottom)

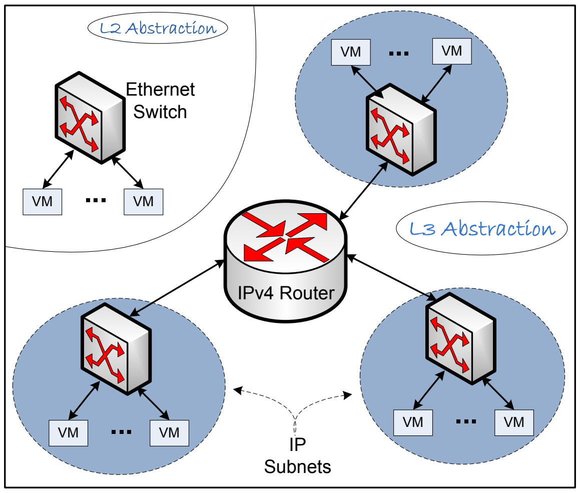


Figure 1: Network abstractions, as seen by the tenants

width and fault tolerance. NetLord leverages our previous work on SPAIN [18] to provide an underlying multi-path fabric using commodity switches. We provide a brief summary of SPAIN in section 3.4.

3.1 A tenant's view of NetLord

NetLord provides tenants with a very simple abstract view of the network: every tenant has one or more *private* MAC address spaces. In each of its MAC address spaces, the tenant can assign arbitrary MAC addresses. A tenant might wish to use multiple MAC address spaces to simplify address allocation, or to limit the scope of its broadcasts/multicasts. (NetLord does not currently implement tenant multicasting, but we believe this is feasible.)

Most tenants will also allocate and use L3 addresses (IPv4, IPv6, etc.). NetLord mostly ignores any tenant-visible L3 issues (except as discussed in sections 3.2 and 3.9). Therefore, by placing no restrictions on how tenants assign L2 or L3 addresses, NetLord provides full address-space virtualization: multiple tenants can use the same address without having their packets mis-routed.

3.2 Virtual routing

A tenant can divide its IP address space into networks and/or IP subnets, and connect these via software routers running on some of its VMs. This approach requires no support from NetLord.

However, for simple routing functions, the extra network hop and VM computation implied by that approach can add unnecessary overhead. Therefore, NetLord follows Diverter's model of supporting "virtual routing" within the hypervisor [11]. A tenant designates to NetLord certain sets of ⟨IP address, MAC address⟩ pairs (within their own address spaces) as virtual router interfaces. Whenever a tenant VM sends a packet to one of these MAC addresses, its local NetLord agent intercepts the packet, extracts the IP header, and does a route lookup to determine the destination tenant-assigned IP and MAC addresses (see section 3.9 for the latter). The NLA can then encapsulate the outgoing packet to send it directly to the final destination.

The virtual routing function could support simple firewall functions, although tenant-implemented SW routers might be needed for more complex router features. Tenants can also use virtual routing to exchange packets with hosts on the public Internet, or with other tenants, via a *public address space* that is exposed to all tenants and advertised externally. This public address space is associated with a reserved *Tenant_ID*=2, so the NLA allows any tenant to request that a VM's interface be given an address in that address space.

Figure 1 shows examples of several network abstractions available to NetLord tenants. The inset shows a pure L2 abstraction; the main figure shows a tenant with three IPv4 subnets connected by a virtual router.

3.3 NetLord's components

Figure 2 shows a high-level architectural view of NetLord. The top half of the figure depicts important components and their interconnections, while the lower half shows the header operations performed on a packet as it travels through these components.

As shown in the top half, NetLord consists of: (1) a fabric consisting of simple switches, (2) NetLord Agents (NLAs) in the hypervisor at each physical host, and (3) a configuration repository.

Fabric switches: NetLord relies on a traditional, switched Ethernet fabric, using unmodified commodity switches. We require only that these switches support VLANs (for multi-pathing; see section 3.4) and basic IP forwarding. We do not require full-fledged support for IP routing; in particular, the switches run no routing protocol. We do require that a switch can take an IP packet sent to its own MAC address, look up its destination using longest-prefix match (LPM) in a small forwarding table using statically-configured entries, and forward the packet appropriately. All of the datacenter switches we have examined, including the cheapest ones, can support NetLord.

These switches tend to be much cheaper than full routers, because they do not require support for complex routing protocols (e.g., IS-IS or OSPF), large routing tables, complex firewall functions, etc.

NetLord agents: A NetLord Agent (NLA) resides in the hypervisor (or the driver domain) of each physical server, and performs two major tasks. First, the NLA transparently encapsulates and decapsulates all packets from and to the local VMs.

Second, the NLA collaborates with other NLAs, and with the central configuration repository, to gather and maintain all the information needed for the encapsulation. For instance, the NLA builds and maintains a table that maps a VM's Virtual Interface (VIF) ID to the port number and MAC address of the edge switch to which the server hosting that VM is connected.

Configuration repository: The repository (which could be replicated for performance and availability) resides at an address known to the NLAs, and maintains several databases. Some are used for SPAIN-style multi-pathing; some are used for per-tenant configuration information. We envision this repository to be co-located with the datacenter-wide VM manager system (such as Eucalyptus[1]) that we expect all cloud datacenters to have. (Note that this repository differs from OpenFlow's central controller, since it is used to configure end-host parameters, not just switches).

3.4 SPAIN in a nutshell

NetLord relies on SPAIN to construct a high-bandwidth, resilient multi-path fabric using commodity Ethernet switches. We briefly sketch the pertinent features; for a full description, see [18].

SPAIN is based on three mechanisms: it pre-computes k edge-disjoint paths between pairs of edge switches; it pre-configures VLANs to identify these paths (not for isolation, as is the typical use of VLANs); and it uses an end-host agent to spread the traffic across paths (i.e., VLANs).

SPAIN's algorithms for computing edge-disjoint paths, merging these paths into trees, and optimally packing these into the 12-bit VLAN tag space are complex and somewhat computationally expensive, but are run only when the network topology is designed or significantly changed. These algorithms work with any topology, but are most useful when the topology provides a rich variety of paths (e.g., FatTree).

SPAIN uses an end-host agent, which can be incorporated into a hypervisor, and thus integrated with the NetLord Agent. On packet transmission, the agent looks up the destination Ethernet address D in a local table, yielding a set of k VLANs that reach that destination. It then chooses a VLAN (e.g., round-robin, but with flow affinity to prevent packet reordering), tags the packet with that VLAN, and transmits it normally. Because the k VLANs are constructed to take different paths to D, this provides high net bandwidth and load balancing among paths. The SPAIN agent also detects failed $\langle$VLAN, destination$\rangle$ pairs, and then re-routes around the failure by using a different VLAN.

SPAIN by itself (i.e., without NetLord) suffers from a major scalability problem: it not only exposes the fabric switches to end-host MAC addresses, but it exposes each VM MAC address k times: once per $\langle$VLAN, VM-MAC$\rangle$ pair. This means that SPAIN creates even more pressure on switch data-plane FIB tables than plain Ethernet does. (We demonstrate this problem in section 5.) However, NetLord encapsulates all tenant VM addresses, and also hides most physical NIC MAC addresses from most switches (as we will soon describe). Thus, augmenting SPAIN with NetLord greatly reduces FIB pressure, because the switch FIBs need to hold only one entry for each $\langle$VLAN, switch-MAC$\rangle$ pair, and there are far fewer switches than VMs.

The SPAIN end-host agent relies on the repository to obtain the table that maps between destinations and sets of VLANs. When combined with the NetLord Agent, SPAIN requires one entry for each edge switch in the datacenter (not for each VM!); this table is loaded at boot time and updated only when the set of switches and wires changes. (Note that the NLAs on each edge switch will see a different table; the NLAs attached to one switch all see the same table.)

3.5 Encapsulation details

NetLord identifies a tenant VM's Virtual Interface (VIF) by the 3-tuple $\langle$Tenant_ID, MACASID, MAC-Address$\rangle$, where MACASID is a tenant-assigned MAC address space ID, to support the use of multiple L2 address spaces.

When a tenant VIF *SRC* sends a packet to a VIF *DST*, unless the two VMs are on the same server, the NetLord Agent must encapsulate the packet. The encapsulation is constructed as follows (and as depicted in the lower half of figure 2):

VLAN.tag	=	SPAIN_VLAN_for(edgeswitch(DST),flow)
MAC.src	=	edgeswitch(*SRC*).MAC_address
MAC.dst	=	edgeswitch(*DST*).MAC_address
IP.src	=	MACASID
IP.dst	=	encode(edgeswitch_port(*DST*), Tenant_ID)
IP.id	=	same as VLAN.tag
IP.flags	=	Don't Fragment

This encapsulation conveys all of the information that the receiving NLA needs to deliver the packet, since the Tenant_ID is encoded in the IP.dst field, the MACASID is carried in the IP.src field, and the original packet header contains the MAC-Address of the destination VIF. Because the sender NLA sets the "Don't Fragment" flag, it can use the IP.id field to transmit the SPAIN-assigned VLAN tag to the receiving NLA. The outer VLAN tag is stripped by the egress edge switch, but the receiving NLA needs to see this tag to support SPAIN's fault-tolerance mechanisms; see section 3.8.

Clearly, the NLA needs to know the remote edge switch MAC address, and the remote edge switch port number, for the destination VM; we will explain how this happens in section 3.9. (Section 3.7 explains how the NLA learns the local edge switch MAC address.) It also must use SPAIN to choose a VLAN for the egress edge switch; to avoid packet reordering, this function also takes a (tenant-level 5-tuple) flow ID.

The most subtle aspect of the encapsulation is the function $encode()$, which takes the egress edge switch port number p (for the destination server, represented as a 7-bit number) and the Tenant_ID tid to create an IP address. Our goal is to allow the egress edge switch to look up this address to generate the final forwarding hop. We construct this encoding by creating an IP address with p as the prefix, and tid as the suffix: $p.tid[16{:}23].tid[8{:}15].tid[0{:}7]$, which supports 24 bits of Tenant_ID. (Other similar encodings are possible, if, for example, the edge switches have more than 128 ports.) Note that since these addresses are never used outside the context of the egress edge switch, we can use any properly formed IP address.

This address encoding allows the egress edge switch to use longest-prefix matching, over a forwarding table with wildcard addresses of the form $p.*.*.*/8$ to do the final-hop lookup. This table needs only one entry per local switch port, and so will fit on

[1]http://eucalyptus.cs.ucsb.edu/

the cheapest switches. Another key point is that this table is also the same on every edge switch.

In summary, the NetLord encapsulation uses the destination MAC address to cover all VMs attached to a single edge switch, and it uses the destination IP address both to direct the packet to the NLA on the correct server, and to allow that NLA to deliver the packet to the correct tenant.

3.6 Switch configuration

Switches require only static configuration to join a NetLord network. These configurations are set at boot time, and need never change unless the network topology changes.

The key idea behind edge-switch configuration is that there is a well-defined set of IP addresses that are reachable via a given switch port p: $encode(p, *)$, or more concretely, p.*.*.*/8. Therefore, when an edge switch boots, either a local process or a management station simply creates IP forwarding-table entries of the form $\langle prefix,\ port,\ next\_hop\rangle = \langle p.*.*.*/8,\ p,\ p.0.0.1\rangle$ for each switch port p. The NLA on each server "owns" the next-hop IP address $p.0.0.1$, as described in section 3.7.

These forwarding-table entries can be set up by a switch-local script, or a management-station script via SNMP or the switch's remote console protocol. Since, as noted above, every edge switch has exactly the same IP forwarding table, this simplifies switch management, and greatly reduces the chances of misconfiguration.

All NetLord switches also need to be given their SPAIN VLAN configurations. This information comes from the configuration repository, and is pushed to the switches via SNMP or the remote console protocol. When there is a significant topology change, the SPAIN controller might need to update these VLAN configurations, but that should be a rare event.

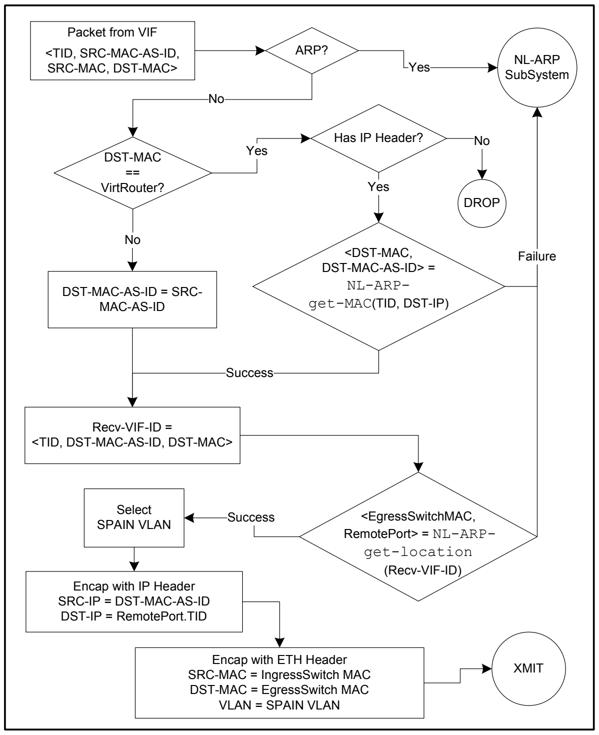
3.7 NetLord Agent configuration

NetLord Agents need several kinds of configuration information. We defer discussion, until section 3.9, of how an NLA learns information about where remote tenants are in the network. We also assume that there already is a VM manager that places VMs on hosts, and so is already distributing VM configuration information (including tenant IDs) to the hypervisors. Finally, we assume that when a tenant VM creates a VIF, the VIF parameters become visible to the hypervisor.

The NLA must learn its own IP address and edge-switch MAC address. When a NetLord hypervisor boots, it listens to the Link Layer Discovery Protocol (LLDP - IEEE 802.1AB) messages sent by the edge switch to which it is connected. An LLDP message tells the server the switch's port number p and MAC address. The NLA then assumes the IP address $encode(p, 1) = p.0.0.1$ as its own, and responds to ARP queries for that address from the local edge switch.

If a server has multiple interfaces, the NLA repeats this process on all of them. Multiple interfaces of the same server could end up with the same IP address, because they could be connected to the same-numbered port on different switches. This is not a problem: the NetLord agent never actually uses these IP addresses, except to respond to ARP requests, which can be handled locally to a specific interface.

Since the $p.0.0.1$ address has no meaning beyond the local switch, an NLA needs a globally-usable address for communication with repositories, VM managers, and other hypervisors. NetLord reserves an address space for Tenant_ID=1, and each NLA must obtain an IP address in this address space. Therefore, when the hypervisor boots, it broadcasts a DHCP request directly over the L2 fabric, using its hardware MAC address, and the DHCP server responds with sufficient information for the NLA to continue with further operation in the Tenant_ID=1 space. (While this bootstrap mechanism does require L2 switches to learn the server's MAC address, this is needed only briefly, and so does not create much FIB pressure.)



Packet from VIF
<TID, SRC-MAC-AS-ID, SRC-MAC, DST-MAC>
ARP?
Yes
NL-ARP SubSystem
No
DST-MAC == VirtRouter?
Yes
Has IP Header?
No
DROP
Yes
No
<DST-MAC, DST-MAC-AS-ID> = NL-ARP-get-MAC(TID, DST-IP)
Failure
DST-MAC-AS-ID = SRC-MAC-AS-ID
Success
Recv-VIF-ID = <TID, DST-MAC-AS-ID, DST-MAC>
Select SPAIN VLAN
Success
<EgressSwitchMAC, RemotePort> = NL-ARP-get-location (Recv-VIF-ID)
Encap with IP Header SRC-IP = DST-MAC-AS-ID DST-IP = RemotePort.TID
Encap with ETH Header SRC-MAC = IngressSwitch MAC DST-MAC = EgressSwitch MAC VLAN = SPAIN VLAN
XMIT

Figure 3: Flow Chart for Packet Send

3.8 The journey of a NetLord packet

We can now describe how a packet flows through the NetLord system.

VM operation: VMs send packets out exactly as they would have without NetLord, because NetLord is fully transparent to VMs. An outbound packet thus arrives at the local NLA.

Sending NLA operation: The flow chart in Figure 3 shows packet processing within the sending NLA. ARP messages are handled by NL-ARP subsystem (section 3.9). For all other packets, the first step is to determine the unique ID of the packet's destination Virtual Interface (VIF), ⟨Tenant_ID, MACASID, MAC-Address⟩.

The Tenant_ID of the receiving VIF is always the same as that of the sender VIF; NetLord does not allow direct communication between two tenants, except via the public address space with Tenant_ID=2. The NLA therefore needs to determine the other two fields of the VIF: MACASID and MAC-Address.

If the L2 packet is *not* addressed to the MAC of a designated virtual router within the VIF's MACASID (see section 3.2), then the destination VIF's MACASID must be the same as the source VIF's MACASID, and the destination VIF's MAC-Address can be found in the L2 packet header. A tenant that wants to move packets between two different MAC address spaces can do this by using a VM with VIFs in either address space as a software router.

If the packet *is* MAC-addressed to the virtual router, the NLA extracts the destination IP address from the packet, then obtains the destination VIF's ⟨MACASID, MAC-Address⟩ directly from the NL-ARP lookup table (which, because it also supports ARP-like functions, associates an IP address with a VIF, if that binding exists).

Once the NLA has the destination VIF ID, it must determine the destination edge-switch MAC address and server port number. This done by lookup in a table maintained by the NL-ARP protocol, which maps from a VIF ID to the correct ⟨MAC-Address, port⟩ tuple.

Once the egress edge switch MAC address is known, we invoke SPAIN's VLAN selection algorithm to select a path for the packet. If the outgoing packet is a TCP packet, NetLord also extracts the 5-tuple and provides it to SPAIN, so that all packets of a given TCP flow take the same path and avoid reordering.

At this point, the NLA has all the information it needs to create an encapsulated packet, using 802.1q and IP headers. The headers are created as described in section 3.5.

If the destination server is attached to the same switch, the NLA sets the source MAC address to its own hardware MAC address, rather than that of the switch. We do this because switches may drop packets if the source and destination MAC addresses are the same. Also, it is OK for the switch to learn the MAC addresses of its directly-connected servers in this case, because these addresses will not leak out to other switches, and the net increase in FIB pressure is limited. In this case, we also do not need to set a SPAIN VLAN tag, because there is only one possible path.

Network operation: The packet follows a path, defined by the SPAIN-provisioned VLAN, through the switched L2 network. All the switches *en route* learn the reverse path to the ingress edge switch, because its MAC address is carried in the packet as the source MAC.

On receiving the packet, the egress edge switch recognizes the destination MAC as its own, strips the Ethernet header, and then looks up the destination IP address in its IP forwarding table to determine the destination NLA next-hop information, which (by the construction in section 3.5) gives the switch-port number and local IP address of the server. The switch might occasionally need to do an ARP request to find the server's MAC address.

Receiving NLA operation: The receiving NLA decapsulates the MAC and IP headers, after extracting the Tenant_ID from the encapsulating IP destination address, the MACASID from the IP source, and the VLAN tag from the IP_ID field. It can then use its local information to look up the correct tenant's VIF, using the L2 destination address in the inner packet. (A tenant might have multiple VMs on a server, but each has a unique VIF.) Finally, it delivers the packet to the tenant VIF.

The NLA also notifies the local SPAIN agent that a packet was received from the ingress edge switch MAC address, via the copy VLAN tag carried in the IP.ID field. SPAIN needs this information to monitor the health of its paths. (The original SPAIN VLAN tag in the 802.1q header has already been stripped by the egress edge switch, so we need this copy to convey the tag to the receiving NLA.)

3.9 NL-ARP

An NLA needs to map a VIF to its location, as specified by an egress switch MAC address and port number. This mapping combines the functions of IP routing and ARP, and we use a simple protocol called NL-ARP to maintain an *NL-ARP table* in each hypervisor. The NLA also uses this table to proxy ARP requests made by the VMs, rather than letting these create broadcast load.

NL-ARP defaults to a push-based model, rather than the pull-based model of traditional ARP: a binding of a VIF to its location is pushed to all NLAs whenever the binding changes. In section 4.2, we argue that the push model reduces overhead, and simplifies modelling and engineering the network.[2]

The NL-ARP protocol uses three message types: NLA-HERE, to report a location; NLA-NOTHERE, to correct misinformation; and NLA-WHERE, to request a location. Messages can either be broadcast on the L2 fabric, over a special VLAN that reaches all NLAs, or unicast to one NLA, through the NetLord mechanism via Tenant_ID=1. NL-ARP broadcasts are sent using the ingress switch MAC address as the source address, to avoid adding FIB pressure.

When a new VM is started, or when it migrates to a new server, the NLA on its current server broadcasts its location using an NLA-HERE message. Since NL-ARP table entries are never expired, in the normal case a broadcast is needed only once per VM boot or migration.

Broadcasts can be lost, leading to either missing or stale entries. If an entry is missing when a tenant sends a packet, the sending NLA broadcasts an NLA-WHERE request, and the target NLA responds with a unicast NLA-HERE. If a stale entry causes misdelivery of a packet, the receiving NLA responds with a unicast NLA-NOTHERE, causing deletion of the stale entry, and a subsequent NLA-WHERE broadcast. (Entries might also be missing after the unlikely event of a table overflow; see section 4.2.)

4. BENEFITS AND DESIGN RATIONALE

Having described NetLord's architecture and operation, we now explain its benefits, and the rationale behind some of the important design decisions. We first explain how NetLord meets the goals in Section 2.1. We then discuss some design alternatives we considered, and explain how we derived our architecture, subject to the restrictions imposed by our goals.

4.1 How NetLord meets the goals

Simple and flexible abstractions: NetLord gives each tenant a simple abstract view of its network: all of its VMs within a MAC address space (e.g., within an IP subnet) appear to be connected via a single L2 switch, and all IP subnets appear to be connected via a single virtual router. The switch and router scale to an arbitrary number of ports, and require no configuration aside from ACL and QoS support in the virtual router. The tenant can assign L2 and L3 addresses however it chooses, and for arbitrary reasons. NetLord's address space virtualization therefore facilitates development of novel network and transport protocols within a cloud datacenter.

Scale: Number of tenants: By using the p.*.*.*/24 encoding (see section 3.5), NetLord can support as many as 2^{24} simultaneous tenants. (We could increase this, if necessary, by using additional header fields for the encoding, at the cost of added complexity.)

Scale: Number of VMs: NetLord's encapsulation scheme insulates the L2 switches from all L2 addresses except those of the edge switches. Each edge switch also sees a small set of locally-connected server addresses. The switches are thus insulated from the much larger number of VM addresses, and even from most of the server addresses.

Because this limits FIB pressure, a NetLord network should scale to a huge number of VMs, but it is hard to exactly quantify this number. The actual number of FIB entries required depends upon a complex interaction of several factors that are either poorly understood, that vary widely over small time scales, or that are hard to quantify. These factors include the traffic matrix and its locality;

[2]An alternative to NL-ARP is to use a DHT, as in SEATTLE [17].

packet and flow sizes, application tolerance of lost packets; network topology; dynamics of load balancing schemes; etc.

Instead, we estimate the number of unique MAC addresses that NetLord can support, based on a set of wildly pessimistic assumptions. We assume a FatTree topology (these scale well and are amenable to analysis), and we assume a worst-case traffic matrix: simultaneous all-to-all flows between all VMs, with one packet per flow pending at every switch on the flow's path. We assume that the goal is to never generate a flooded packet due to a capacity miss in any FIB.

Table 1: NetLord worst-case limits on unique MAC addresses

Switch	FIB Sizes			
Radix	16K	32K	64K	128K
24	108,600	153,600	217,200	307,200
48	217,200	307,200	434,400	614,400
72	325,800	460,800	651,600	921,600
94	425,350	601,600	850,700	1,203,200
120	543,000	768,000	1,086,000	1,536,000
144	651,600	921,600	1,303,200	1,843,200

Based on these assumptions, NetLord can support $N = VR\sqrt{(F/2)}$ unique MAC addresses, where V is the number of VMs per physical server, R is the switch radix, and F is the FIB size (in entries). (SPAIN requires a FIB size proportional to the square of the number of edge switches.)

Table 1 shows the maximum number of MAC addresses (VM) that NetLord could support, for various switch port counts and FIB sizes. Following a rule of thumb used in industrial designs, we assume $V = 50$.

With existing single-chip switches that feature 72 ports and 64K FIB entries [12], a NetLord network could support 650K VMs. Using plausible future 144-port ASICs with the same FIB size, NetLord could support 1.3M VMs. The table suggests that ASIC designers might consider increasing port count at the expense of FIB size; the number of VMs scales linearly with ports, but only with the square root of the FIB size.

Note that the original SPAIN design can scale to large numbers of physical paths, but at the cost of large FIBs. NetLord removes this restriction, and so allows the network to support a number of VMs consistent with its physical scale.

Ease of operation: NetLord simplifies network operation in three different ways, thereby reducing operating costs. First, as explained in Section 3, NetLord automates all switch configuration. The configuration details are either computed offline (e.g., SPAIN's VLAN configuration), or are autonomously determined by individual entities (e.g., the IP forwarding tables in the switches). Also, most of the configuration is static; this not only eliminates the need for constant supervision by humans, but also makes debugging and trouble-shooting easier.

Second, NetLord makes it easy to exploit standard mechanisms on commodity switches to implement tenant-specific traffic engineering, ACL, and isolation policies. As a design principle, NetLord uses only standard header formats; this exposes all important tenant information in header fields supported by even the most basic ACL mechanisms. For instance, an ACL based on the low-order bits of the destination IP address (the tenant ID) can match all packets belonging to a single tenant. Similarly, a tenant's flow between two physical servers can be easily identified by matching on the source and destination MAC addresses, and on the port-number bits of the IP destination address. One could use these mechanisms, together with NLA support, to deploy sophisticated QoS via a central controller (analogous to OpenFlow).

Finally, by supporting high bisection bandwidth without concerns about FIB pressure, NetLord removes network-based restrictions on VM placement. The VM manager need not try to co-locate communicating VMs, and can instead place them based on resource availability (e.g., CPU or RAM) or power management.

Low-cost switches: NetLord can efficiently utilize inexpensive, feature- and resource-limited commodity switches. The datacenter operator need not pay for IP routing support, or to upgrade switches to support novel L3 features (e.g., IPv6).

4.2 Design rationale

Why encapsulate? VM addresses can be hidden from switch FIBs either via encapsulation or via header-rewriting. Encapsulation is often thought to suffer three major drawbacks: (1) Increased per-packet overhead – extra header bytes and CPU for processing them; (2) Heightened chances for fragmentation and thus dropped packets; and (3) Increased complexity for in-network QoS processing. Even so, we used encapsulation for two reasons. First, the header re-writing cannot simultaneously achieve both address-space virtualization and reduced FIB pressure. Second, in a datacenter, we can easily address the drawbacks of encapsulation.

Header rewriting clearly would not suffer from increased byte overheads or fragmentation. However, since we want to support an unconstrained L2 abstraction, and therefore cannot assume the presence of an IP header, the only two fields we could rewrite are the source and destination MAC addresses. But to avoid FIB pressure from exposed VM addresses, we would have to rewrite these header fields with the source and destination agent (server) MAC addresses; this would then make it impossible for the receiving agent to know which VM should receive the packet. Diverter [11] solves this problem by requiring tenant VMs to send IP packets (using a defined addressing scheme), which allows the agent to map the destination IP address to the correct VM. Also, this approach still exposes server MAC addresses to switch FIBs, which limits scalability. (Nor could we rewrite using edge-switch MAC addresses, because the egress edge switch would not know where to forward the packet.) Thus, we believe encapsulation is required to offer tenants an L2 abstraction.

What about per-packet overheads, both on throughput and on latency? Our measurements (section 5) show that these overheads are negligible. Modern multi-GHz CPUs need only a few tens of nanoseconds to add and delete the extra headers; the added wire delay at 10Gbps is even smaller. Throughput overheads should be negligible if tenants use 9000-byte "jumbo" packets, which all datacenter switches support.

Encapsulation in NetLord also adds one extra hop to Diverter's "one-hop" paths. However, NetLord suffers no performance loss, because unlike in Diverter, this extra hop is done in hardware of the egress edge switch.

We can also mitigate fragmentation by exploiting jumbo packets. Our NetLord virtual device (in the hypervisor) exports a slightly smaller MTU (34 bytes fewer) than the physical device. Because in a datacenter we can ensure a consistent, high MTU across all switches, and our encapsulation header always sets the "Do-not-fragment" bit, we completely eliminate the possibility of NetLord-generated fragments. (This also frees up the fragmentation-related IP header fields for us to use for other purposes; see section 3.5.)

Finally, as explained in section 4.1, we can exploit NetLord's encapsulation scheme to make it easier to identify tenants via simple ACLs.

Why not use MAC-in-MAC? We could have used a "MAC-in-MAC encapsulation", where the sending NLA encapsulates the packet with only a MAC header, setting the source and destination MAC addresses set to those of the ingress and egress switches. This would add less overhead than our chosen encapsulation, but would create some problems. First, this would require the egress edge-switch FIB to map a VM destination MAC address to one of its output ports. This implies that no two VMs connected to an edge switch could have the same MAC address, which would impose unwanted restrictions on VM placement. Further, since arbitrary tenant-assigned L2 addresses, unlike NetLord's encoded IP address, cannot be aggregated, this approach requires an order of magnitude more FIB entries. Perhaps worse, it also requires some mechanism to update the FIB when a VM starts, stops, or migrates. Finally, MAC-in-MAC is not yet widely supported in inexpensive datacenter switches.

NL-ARP overheads: NL-ARP imposes two kinds of overheads: bandwidth for its messages on the network, and space on the servers for its tables.

In steady-state operation, most NL-ARP messages would be the gratuitous NLA-HERE broadcasts. We show, through a simple analysis, that these impose negligible overheads.

All NLA messages fit into the 64-byte (512-bit) minimum-length Ethernet packet. With the worst-case 96-bit inter-frame gap, an NLA packet requires 608 bit-times.

Most new servers come with at least one 10Gbps NIC. If we limit NLA-HERE traffic to just 1% of that bandwidth, a network that supports full bisection bandwidth can sustain about 164,000 NLA-HERE messages per second. That is, the network could support that rate of VM migrations or boots. Assuming that most VMs persist for minutes or hours, that should be sufficient for millions of VMs. Further, we believe that NLA-WHERE broadcasts and NLA-NOTHERE unicasts should occur so rarely as to add only negligible load.

NL-ARP also requires negligible table space on servers. Each table entry is 32 bytes. A million such entries can be stored in a 64MB hash table, at 50% loading. Since even small datacenter servers have 8GB of RAM, this table consumes about 0.8% of that RAM.

5. EXPERIMENTAL ANALYSIS

We did an experimental analysis, both to measure the overhead of the NetLord Agent (NLA), and to evaluate the scalability of NetLord. We measured overheads using a micro-benchmark (section 5.1) and scalability by emulating thousands of tenants and hundreds of thousands of VMs. We have not yet measured the cost of the control plane, including NL-ARP.

We implemented the NetLord agent as a Linux kernel module. We started with our implementation of SPAIN [18], adding about 950 commented lines of code. This includes all components of NetLord, except the NL-ARP subsystem. We implemented NL-ARP lookup tables, but have not fully implemented the message handlers, so we ran our experiments using statically-configured lookup tables; thus, no NL-ARP messages are sent. We ran our tests using Ubuntu Linux 2.6.28.19.

5.1 NetLord Agent micro-benchmarks

Since the NLA intercepts all incoming and outgoing packets, we quantified its per-packet overheads using two micro-benchmarks, "ping" for latency and Netperf [3] for throughput. We compare our NetLord results against an unmodified Ubuntu ("PLAIN") and our original SPAIN implementation. We used two Linux hosts (quad-core 3GHz Xeon CPU, 8GB RAM, 1Gbps NIC) connected via a pair of switches that served as "edge switches."

Table 2: Microbenchmarks for NetLord overheads

Case	Metric	PLAIN	SPAIN	NetLord
Ping	avg	97	99	98
(in μs)	min/max	90/113	95/128	93/116
NetPerf	avg	987.57	987.46	984.75
1-way	min	987.45	987.38	984.67
(in Mbps)	max	987.67	987.55	984.81
NetPerf	avg	1835.26	1838.51	1813.52
2-way	min	1821.34	1826.49	1800.23
(in Mbps)	max	1858.86	1865.43	1835.21

Our ping (latency) experiments used 100 64-byte packets. The first row in Table 2 shows the results (average, minimum, and maximum). NetLord appears to add at most a few microseconds to the end-to-end latency.

We measured both one-way and bi-directional TCP throughput using Netperf. For each case, we ran 50 10-second trials. We used jumbo (9000-byte) packets, because we observed erratic results (even with PLAIN) when using 1500-byte packets in the two-way experiments. (We hypothesize, but have not confirmed, that this is due to the limited number of buffer descriptors in the specific NIC in our testbed hosts.) Note that, because of our 34-byte encapsulation headers, NetLord exposes an 8966-byte MTU to applications.

The second and third rows in Table 2 show throughput results (mean, min., and max.). NetLord causes a 0.3% decrease in mean one-way throughput, and a 1.2% drop in in mean two-way throughput. We attribute these nearly-negligible drops mostly to the smaller MTU.

5.2 Emulation methodology and testbed

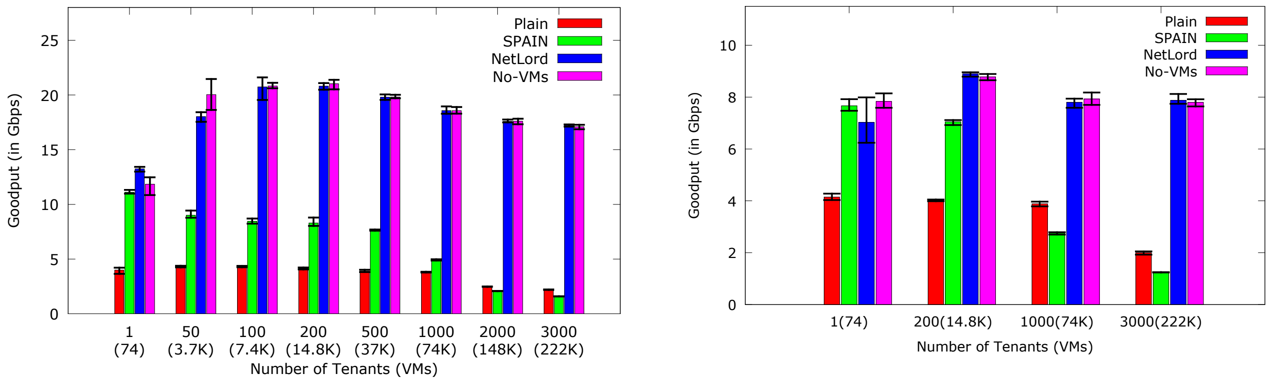
We have asserted that NetLord can scale to large numbers of VMs using inexpensive switches. We were limited to testing on just 74 servers, so to demonstrate NetLord's scalability, we emulated a much larger number of VMs than could normally run on 74 servers.

We implemented a light-weight VM emulation by adding a shim layer to the NLA kernel module. Conceptually, each TCP flow endpoint becomes an emulated VM. The shim layer exposes a unique MAC address for each such endpoint, by mapping from a source or destination $\langle IP\_addr, TCP\_port\rangle$ tuple to synthesize a corresponding MAC address. The shim executes, for every outgoing packet, before the NLA code, and rewrites the packet's MAC source and destination addresses with these synthetic addresses. Thus, the NLA sees one such emulated VM for each flow from our workload-generator application.

Using this technique, we can emulate up to $V = 3000$ VMs per server. We emulated 74 VMs per tenant on our 74 servers (one VM per tenant per host), or $74N$ VMs in all for N tenants.

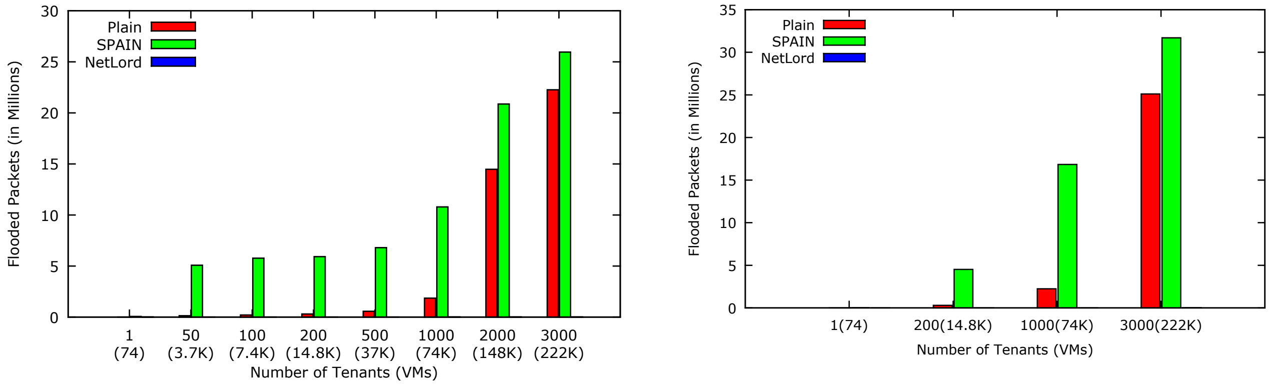
Multi-tenant parallel shuffle workload: Our workload generator emulates the *shuffle* phase of Map-Reduce. Each of the N tenants has 74 VMs, each emulating both a Map task and a Reduce task. Each Map task transfers 10MB to each of the tenant's 73 other Reduce tasks. A given Map tasks serializes these transfers, in a random order that differs for each tenant; the transfers originating from a single Map task do not run in parallel. To avoid overloading a Reduce task, we reject incoming connections beyond a limit of 3 simultaneous connections per Reduce task. A Map task

[3] www.netperf.org



(a) FatTree Topology (b) Clique Topology

Figure 4: Goodput with varying number of tenants (number of VMs in parenthesis).



(a) FatTree Topology (b) Clique Topology

Figure 5: Number of flooded packets with varying number of tenants (number of VMs in parenthesis).

will postpone any such rejected transfer to a random position in its pending-transfer list; ultimately, all such tasks will complete.

Between each trial of the entire workload, we delay for 320 seconds, during which no packets are sent. Since our switches are set to expire learning table entries after 300 seconds, this ensures the FIBs are empty at the start of each trial.

Metrics: We run each shuffle trial either to completion or for 1800 seconds, and we compute the goodput as the total number of bytes transferred by all tasks, divided by the run-time. We also measure the number of unnecessarily-flooded packets during a trial, by counting, at each host, the total number of packets received that are not destined for any VM on that host.

Testbed: We ran the emulation experiments on 74 servers, part of a larger shared testbed. Each server has a quad-core 3GHz Xeon CPU and 8GB RAM. The 74 servers are distributed across 6 edge switches, with 10, 12, 12, 13, 13, and 15 servers/switch. All switch and NIC ports run at 1Gbps, and the switches can hold 64K FIB table entries.

We built two different topologies on this testbed: (i) **FatTree topology**: the entire testbed has 16 edge switches; using another 8 switches to emulate 16 core switches, we constructed a two-level FatTree, with full bisection bandwidth (i.e., no oversubscription). (ii) **Clique topology**: All 16 edge switches are connected to each other, thus creating a clique, which is oversubscribed by at most 2:1.

SPAIN and NetLord exploit these multi-path topologies by using VLANs to identify paths. For FatTree, we used 16 VLANs, each rooted at one core switch. For Clique, we used 6 VLANs, each rooted at one of the edge switches involved in our experiments.

5.3 Emulation results

As with the micro-benchmarks, we compared PLAIN, SPAIN, and NetLord. PLAIN does not support multi-pathing, and so its traffic follows a single spanning tree. We ran trials on both of the topologies with varying numbers of tenants (N), and for each N, we ran at least three trials and computed the means.

Figures 4(a) and 4(b) show the mean goodputs for the FatTree and Clique topologies, respectively. We include error bars showing the maximum and minimum results. The x-axis legend shows both the number of tenants, and the number of emulated VMs in parentheses.

Figure 4 shows that:

(1) As expected, PLAIN does not scale. Aside from failing to exploit the available bisection bandwidth, PLAIN's goodput drops after the number of VMs exceeds the switch FIB table size (64K entries) – by as much as 44% for 222K VMs. (At 74K VMs, the drop is only 3%, implying that the FIB table replacement policy might be LRU.)

(2) SPAIN outperforms PLAIN for <74K VMs, since it does exploit the multi-path fabrics.

(3) However, SPAIN performance degrades badly as the number of VMs increases. Above 74K VMs, SPAIN actually performs worse than PLAIN, because it maintains k distinct end-to-end paths between host pairs. Thus, instead of requiring $74N$ FIB table entries for N tenants, it requires $74kN$ entries (in the worst case), because a switch maintains a FIB entry for each $\langle$MAC-address,VLAN$\rangle$ tuple. SPAIN's goodput therefore drops even at relatively modest numbers of VMs.

(4) NetLord scales well, achieving far superior goodput, especially as the number of VMs increases.

We observe that NetLord's goodput declines slightly as the number of VMs exceeds 14.8K. We suspect this dip is caused by end-host overheads associated with maintaining lots of TCP connections. To validate this hypothesis, we re-ran the shuffle workload without VM emulation, using the PLAIN system. The goodput we achieved with this case (shown in Figure 4 s "No-VMs", with the number of connections matching the number of VMs) suffers from a similar dip – that is, NetLord's goodput closely matches the No-VMs goodput. The one exception, for Clique and $N = 1$, might be due to NetLord's slightly higher overheads.

Why does NetLord scale better? Our main hypothesis in this paper has been that PLAIN and SPAIN do not scale because of FIB-table misses. Our switches do not allow us to directly count these misses, so we use our counts of flooded packets as a proxy.

Figure 5 shows the mean number of packet floods received at each host during our experiments. (The error bars for this figure are too narrow to be worth showing, and we omit them to avoid clutter). Clearly, PLAIN and SPAIN suffer from lots of flooding, supporting our belief that their scaling problems result from FIB-table capacity misses. Since NetLord conceals all VM and most server MAC addresses from the switches, it experiences only a modest number of FIB misses, caused mostly by table timeouts.

We note that NetLord suffers from more misses than PLAIN when there is only one tenant, although the effect is too insignificant to see in Figure 5 – for FatTree, for example, PLAIN floods about 2,550 packets, SPAIN floods about 25,600, while NetLord floods about 10,500 packets. In the one-tenant case, we see more flooded packets than we expect during the first few connections. This could be because our switches implement their MAC learning process in control-plane software, and some FIB table updates might not complete within an RTT. However, the results are not fully consistent with this hypothesis, and they require further investigation.

In summary, through experimental evaluation on a real testbed with NetLord prototype, we have demonstrated that NetLord scales to several hundreds of thousands of VMs. We have also shown that it works with unmodified commodity switches.

6. DISCUSSION AND FUTURE WORK

In this section, we discuss how the NetLord architecture addresses several important considerations for operators of large datacenter networks. We also identify several limitations of NetLord, and areas for future work.

Fault-tolerance: An important aspect of scale is the increased likelihood of failures. NetLord directly inherits SPAIN's ability to handle transient network failures. SPAIN monitors the health of its VLAN-based paths by observing all incoming traffic, or by sending special *chirp* packets that serve (among other purposes) as heartbeats. This monitoring allows SPAIN (and therefore NetLord) to quickly detect failed paths, and to re-route load to alternate paths.

NetLord does not proactively monitor the health of the servers themselves. NetLord tolerates server failures, because there is no "hard state" associated with a specific NLA. We assume that a VM manager (such as Eucalyptus) is responsible for detecting failed servers and restarting affected VMs on other servers. When the VMs are restarted, their local NLAs broadcast NLA-HERE messages, which repopulate the NL-ARP tables at the NLAs for other VMs. It might be useful for the VM manager to provide this information directly to the surviving NLAs, thereby avoiding some lost packets during the restart phase.

Broadcasts and multicasts: ARP and DHCP account for a vast majority of broadcasts in today's typical datacenter. Because the NLAs proxy ARP requests, these are never broadcast. We expect the DHCP broadcast load to be similar to the NLA-HERE load, which (in section 4.2) we argued is just a small percentage of network capacity.

However, other tenant multicasts and broadcasts (particularly from malicious, buggy, or inefficient VMs) could become a problem. Scalable techniques for multi-tenant multicasting is a challenge for future work. Possible approaches include rate-limiting at the switches, or mapping broadcasts to a tenant-specific multicast group (as proposed in [11, 13]). But these techniques might not scale, because most switches either treat multicasts as broadcasts, or cannot scale the number of simultaneous multicast-tree pruning sessions to match the number of tenants [23].

Per-tenant management: A cloud provider might wish to manage traffic on a per-tenant basis: for example, to impose bandwidth limits or priorities. While this can often be done in the hypervisors [21], sometimes there might be reasons to exploit features of switch hardware. Doing so requires a way for switches to identify the packets of a specific tenant. If tenants can only be identified by the addresses of their individual VMs, this cannot scale. However, since NetLord exposes the tenant ID as part of the encapsulation's IP destination address, one can use a single ACL per tenant to control features such as rate limiting. This would probably still have to be limited to a small subset of the tenants, given the number of ACLs and rate-limiters supported by commodity switches. Or, the high-order bits of tenant IDs could encode service classes.

Inter-tenant and external communications: Currently, NetLord requires that the tenant implement some sort of routing within its own network, if a VM on its private address space needs to be able to contact the external world. We are currently working a few minor modifications to our basic design, so that we can provide a per-tenant "distributed virtual NAT" service.

Software overheads and SR-IOV: Although NetLord itself is frugal with server resources, the use of hypervisor-based (or driver domain-based) network virtualization imposes substantial overheads [27]. These overheads can be avoided by using Single Root I/O Virtualization (SR-IOV) NICs, which allow VMs direct access to NIC hardware. However, SR-IOV would appear to prevent the NLA from intercepting and encapsulating outgoing packets, and a standard SR-IOV NIC would not know how to decapsulate an incoming NetLord packet. We believe that techniques allowing hypervisors to inject code into the guest-domain drivers [9] could solve the outbound problem; the inbound problem would either involve the hypervisor on every received packet, or would require some modifications to SR-IOV.

Other L2 fabrics: Although NetLord as presented in this paper utilizes SPAIN as the underlying multipathing fabric, it is not closely tied to SPAIN. NetLord can use any fabric that provides an Ethernet abstraction such as TRILL [5], SPB [4], or SEATTLE [17]. NetLord might help with scaling these other solutions. For instance, TRILL switches maintain a FIB entry for each destination MAC address (but mapping the destination MAC to a destination edge switch, rather than to a next-hop port), and so (without NetLord) suffer from the same FIB-pressure problems as traditional switches.

SEATTLE addresses scaling for the core switches. However, SEATTLE's edge switches probably still need relatively large FIB tables, to cache the working set of remote VM MAC addresses; otherwise, table misses incur the significant penalty of traversing the SEATTLE DHT. It would be useful to have a quantified analysis of how well SEATTLE scales in a virtualized environment.

7. CONCLUSIONS

In this paper we presented NetLord, a novel network architecture for multi-tenant cloud datacenters.

Through the encapsulation of a tenant's L2 (Ethernet) packets in its own IP packets, NetLord gains multiple advantages over prior solutions. It allows us to fully virtualize each tenant's L2 and L3 address spaces, which gives the tenants full flexibility in choosing their VM addresses — or not choosing these addresses, if they want to preserve their legacy addressing schemes. Encapsulation also allows NetLord to scale to larger numbers of VMs than could otherwise be efficiently supported using commodity Ethernet switches. NetLord's design can exploit commodity switches in a way that facilitates simple per-tenant traffic management in the network. Our new NL-ARP protocol simplifies the design, by proactively pushing the location information of VMs to all servers. NetLord improves ease of operation by only requiring static one-time configuration that is fully automated.

Our measurements show that the NetLord architecture scales to several thousands of tenants, and hundreds of thousands of VMs. Our experimental evaluation shows that NetLord can achieve at least 3.9X improvement in goodput over existing approaches, and imposes only negligible overheads.

Acknowledgements

We thank our shepherd, Jon Crowcroft, and the anonymous reviewers for their insightful comments. We thank Sujata Banerjee, Anna Fischer, Rick McGeer, and Jean Tourillhes for their help and feedback. Finally, we thank Ken Burden, Pete Haddad, and Eric Wu for their help in setting up the testbed network.

8. REFERENCES

[1] Amazon EC2. `http://aws.amazon.com/ec2/`.

[2] Amazon virtual private cloud. `http://aws.amazon.com/vpc/`.

[3] IEEE 802.1ad - Provider Bridging. `http://www.ieee802.org/1/pages/802.1ad.html`.

[4] IEEE 802.1aq - Shortest Path Bridging. `http://www.ieee802.org/1/pages/802.1aq.html`.

[5] IETF TRILL Working Group. `http://www.ietf.org/html.charters/trill-charter.html`.

[6] Windows Azure. `https://www.microsoft.com/windowsazure/`.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proc. SIGCOMM*, 2008.

[8] M. Arregoces and M. Portolani. *Data Center Fundamentals*. Cisco Press, 2003.

[9] Broadcom Corp. Broadcom Ethernet Network Controller Enhanced Virtualization Functionality. `http://tinyurl.com/4r8vxhh`, Accessed on 1/31/11.

[10] Cisco Catalyst 4500 Series Model Comparison. `http://tinyurl.com/yvzglk`, Accessed on 1/31/11.

[11] A. Edwards, A. Fischer, and A. Lain. Diverter: A New Approach to Networking Within Virtualized Infrastructures. In *WREN 2009*.

[12] Fulcrum MicroSystems. FocalPoint FM6000 ProductBrief. `http://tinyurl.com/4uqvale`, Accessed on 1/31/11.

[13] A. Greenberg, J. Hamilton, and N. Jain. VL2: A Scalable and Flexible Data Center Network. In *Proc. SIGCOMM*, 2009.

[14] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *Co-NEXT*. ACM, 2010.

[15] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In *Proc. SIGCOMM*, 2010.

[16] IEEE 802.1q - Virtual Bridged Local Area Networks. `http://tinyurl.com/5ok4f6`, Accessed on 1/31/11.

[17] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *Proc. SIGCOMM*, pages 3–14, 2008.

[18] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. In *Proc. USENIX NSDI*, 2010.

[19] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proc. SIGCOMM*, 2009.

[20] HP ProCurve 6600 Series. `http://tinyurl.com/4cg8qt9`, Accessed on 1/31/11.

[21] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. In *Proc. 3rd Workshop on I/O Virtualization*, June 2011.

[22] M. Scott, A. Moore, and J. Crowcroft. Addressing the Scalability of Ethernet with MOOSE. In *Proc. DC CAVES Workshop*, Sept. 2009.

[23] R. Seifert and J. Edwards. *The All-New Switch Book: The Complete Guide to LAN Switching Technology*. Wiley, 2008.

[24] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the Production Network be the Testbed? In *Proc. of OSDI 2010*.

[25] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Seawall: Performance Isolation For Cloud Datacenter Networks. HotCloud'10.

[26] VMware. `http://www.vmware.com`.

[27] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent Direct Network Access for Virtual Machine Monitors. In *Proceedings of HPCA*, 2007.

[28] Xen. `http://www.xen.org`, Accessed on 1/31/11.