Efficient Virtual Network Isolation in Multi-Tenant Data Centers on Commodity Ethernet Switches

Heitor Moraes Marcos A. M. Vieira Ítalo Cunha Dorgival Guedes
Computer Science Department
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
Email: {motta, mmvieira, cunha, dorgival}@dcc.ufmg.br

***Abstract*—Infrastructure-as-a-Service providers need to provision and isolate their tenants's virtual networks. Current network isolation solutions either suffer from limited scalability, incur encapsulation overheads, or require advanced (e.g., OpenFlow) hardware switches. We propose LANES, a system that provides isolation between billions of virtual machines using commodity Ethernet switches without encapsulation overheads. LANES virtualizes each tenant's network address space and configures rules on each server to translate (tenant) virtual addresses to (infrastructure) physical addresses. Virtual address spaces give tenants flexibility when configuring their virtual networks, and physical addresses reduce demand on infrastructure switches. We implement LANES in OpenStack, leveraging OpenStack's network description functionalities and using OpenFlow to configure Open vSwitch on infrastructure servers. Our evaluation shows LANES ensures network isolation with acceptable rule configuration latency.**

I. Introduction

Infrastructure as a Service (IaaS) providers have a growing demand for solutions to allocate and manage the resources offered to their customers (usually called tenants) [1]. Each tenant requires network resources to interconnect a set of virtual machines (VMs) in an arbitrary topology. Ideally, except for specific interconnection agreements, traffic from one tenant's VMs should never be visible to other tenants's VMs; conversely, only that tenant's traffic should be able to reach his VMs. IaaS providers must provision network resources to guarantee isolation between customer networks.

Simple solutions to provision and isolate tenant networks, like Ethernet VLANs, do not scale to large datacenters [2]. Researchers, standards bodies, and industry have proposed several scalable alternatives over the last few years. One common approach is to virtualize the networking infrastructure using tunneling [3], [4], which incurs encapsulation overhead, or to use advanced (e.g., OpenFlow) hardware switches, which results in additional costs.

Our goal is to provide efficient traffic isolation between tenants in a datacenter environment on commodity Ethernet switches. Since each tenant is oblivious to other tenants, tenants's VMs might use incompatible and overlapping network configurations (e.g., using the same IP address). It is the responsibility of the isolation solution to handle this situation and properly deliver packets to the correct VMs.

To achieve this goal we present LANES, a platform to provision virtual networks and ensure traffic isolation on multi-tenant datacenters based on the paradigm of Software Defined Networks (SDN) [5]. LANES provides flexibility and extensibility through standard APIs.

LANES allows IaaS tenants to specify their (layer-2) network topology using OpenStack's network description language [6]. The LANES SDN controller then uses OpenFlow [7] to configure an Open vSwitch instance on each infrastructure server to provision and isolate tenant networks. LANES was designed to be compatible with existing datacenter infrastructures. LANES requires no modification to physical switches; in particular, LANES runs Open vSwitch on infrastructure servers but does not require OpenFlow-enabled network switches.

LANES virtualizes network address spaces and isolates virtual networks using packet rewriting, and does not incur encapsulation overhead. LANES uses a pair of OpenFlow rules at each server's Open vSwitch instance for each virtual link between communicating virtual machines terminating at that server. LANES's address virtualization uses only the physical servers' MAC addresses in the physical network, which avoids scalability issues associated with large layer-2 address domains that arise when virtual machine MAC addresses traverse the physical switches [8]. LANES provides flexibility to IaaS customers and scales to large datacenters while incurring minimum additional costs.

We evaluate our prototype and show that LANES induces negligible additional latency when translating packets to virtualize addresses, and that Open vSwitch rule configuration, which happens only once for each VM pair, takes less than 200ms. We also show that VM traffic under LANES continues responsive and can achieve full bandwidth utilization even when under DoS attacks.

The remainder of this paper is organized as follows. Sec. II presents the idea behind LANES and describes the packet rewriting technique used, while Sec. III discusses the implementation details of our prototype using OpenStack. Sec. IV evaluates the performance of our LANES prototype. Finally, Sec. V discusses related work and Sec. VI presents final remarks and discusses possible future work.

II. LANES

LANES provisions and isolates layer-2 virtual networks in multi-tenant datacenters. A virtual network interconnects virtual machines (VM) that run on multiple hosts. LANES runs a virtual switch on all datacenter servers to intercept packets before they are forwarded. Interception allows LANES to rewrite packets to virtualize network addresses and isolate virtual networks (Sec. II-A). We also present the algorithms

LANES uses to configure packet rewriting and packet forwarding (Sec. II-B). LANES requires no modification to hosted VMs, and puts no restrictions on what IP addresses VMs can use. LANES also works on unmodified commodity learning Ethernet switches and scales to datacenters with millions of VMs and hundreds of thousands of servers. We describe how we implement LANES's address virtualization and packet forwarding in OpenStack on top of existing standards using Open vSwitch, POX, OpenFlow, and Neutron in Sec. III.

A. Network address virtualization using packet rewriting

Current Ethernet switches have tens of thousands of entries in their MAC forwarding tables.[1] A few hundred servers can host tens of thousands of VMs and put significant pressure on switch forwarding tables, causing severe network performance degradation. In an unmanaged Ethernet segment, VMs can spoof Ethernet MAC addresses to perform DoS attacks on switch forwarding tables or sniff traffic.

LANES assigns a unique MAC address to each server and VM in the infrastructure. We denote the MAC address of a server (or VM) x as M_x. When VM u in server s sends a packet to VM w in server r, the virtual switch at server s rewrites the original packet changing M_u to M_s and M_w to M_r. This rewriting allows packets to traverse (from source to destination servers) a physical network made of commodity learning Ethernet switches requiring a single forwarding table entry per server in the infrastructure. No VM MAC ever reaches the network infrastructure, preventing VMs from spoofing MAC addresses. LANES requires a few KB of memory at each server to store mappings from VMs (attached to virtual networks that span that server) to servers and mappings from servers to MAC addresses.

VMs may use multiple and arbitrary IP addresses. If a server hosts VMs with identical (conflicting) IP addresses, the virtual switch needs additional information to forward the packet to the correct destination VM.

LANES associates one flow identifier F_{uw} to the traffic between each pair (u, w) of communicating IPs. Flow IDs are generated on demand when VMs start communicating and need be unique only between pairs of servers (different pairs of servers may use the same flow IDs). As Ethernet switches forward packets based on MAC addresses alone, LANES uses the source and destination IP addresses to store flow identifiers. Although the number of possible flows is up to 2^{64}, servers need to store only one flow ID per pair of communicating IPs. We note LANES creates *one* flow identifier for *all* connections between a pair of communicating IP addresses. This reduces the cases where a new identifier has to be created to when a pair of IP addresses exchange their first packet.

As an example of LANES's address virtualization, consider the deployment scenario in Fig. 1 with two virtual networks, two servers, and six VMs. We consider that servers are configured with address mappings shown in Tab. I (we explain how LANES generates mappings in Sec. II-B) and that VM i uses a single IP address denoted by A_i.

When VM u transmits a packet to VM w, server s's virtual switch intercepts a packet $P = [M_u M_w \mid A_u A_w]$. As this is an outbound packet,[2] LANES looks up the MAC address of VM w's server (this table is not shown), matches packet P's header against line 1 in Tab. I, and rewrites P into $P' = [M_s M_r \mid F_{uw}]$. When the packet enters the physical network, switches learn which port to use to reach M_s and switches that do not yet have a entry for M_r in their forwarding tables broadcast the packet. When server r receives packet P', it identifies this as an inbound packet and looks up F_{uw} in Tab. I to rewrite P' back into P and forward the original packet to VM w.

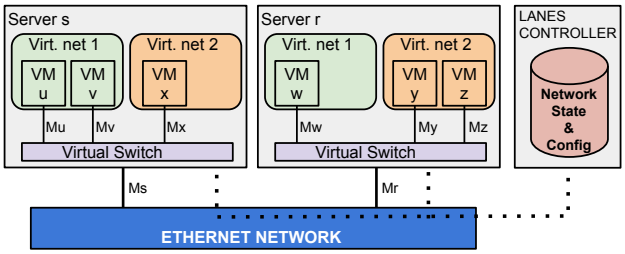


Fig. 1: LANES perspective of an example deployment in a multi-tenant datacenter.

TABLE I: An example flow identifier table for servers in Fig. 1.

OUTBOUND KEY				INBOUND KEY		
SRC MAC	DST MAC	SRC IP	DST IP	FLOW	SRC MAC	DST MAC
M_u	M_w	A_u	A_w	F_{uw}	M_s	M_r
M_w	M_u	A_w	A_u	F_{wu}	M_r	M_s
M_x	M_y	A_x	A_y	F_{xy}	M_s	M_r
M_y	M_x	A_y	A_x	F_{yx}	M_r	M_s

B. Packet forwarding

LANES intercepts all packets that VMs send to perform address virtualization. If a packet matches an entry in the flow identifier table, LANES forwards as indicated by its flow identifier. We now describe how LANES creates flow identifiers for packets with no matches.

LANES requires information about the infrastructure and hosted virtual networks to identify which VMs can communicate and configure flow identifiers accordingly. LANES requires a mapping of VMs to their virtual networks and a mapping of VMs to the infrastructure server where they run. These mappings can be pregenerated according to virtual network configurations or generated on demand as VMs are instantiated, shutdown, or migrated (e.g., when scaling to variable workloads).

LANES also requires a database of which IP addresses are configured in each VM. This database can be pregenerated according to virtual network configurations, or inferred on demand sending ARP packets to all VMs in the tenants's virtual network. Remember that LANES creates one flow identifier for each pair of IPs communicating. To avoid VMs creating arbitrary flow identifiers, the infrastructure provider can limit the number of IPs that a VM can use, or charge for each IP associated with a VM.

In cases where there are no mappings for a packet, LANES determines the packet type, source, and destination. LANES

[1] A Cisco Catalyst 4500-X switch has 55K MAC forwarding table entries.

[2] Inbound and outbound packets can be identified by their input ports, or by checking if the destination MAC address is the server's MAC address.

confirms that source addresses (layers 2 and 3) belong to the sending VM. LANES also checks if source and destination are attached to the same virtual network or if the destination is in an external network (e.g., the Internet). If any of the checks fail, e.g., when VMs are sending packets to non-existing destinations, LANES maps the packet to a special F_{drop} flow identifier. LANES discards all packets that match to F_{drop}. If the packet is to be forwarded, LANES creates a mapping according to the packet's type, source, and destination, as we describe next.

Traffic within a physical server. Authorized traffic between VMs located on the same server is forwarded *without modification* and no packet rewriting is performed. When the first packet of a flow between a pair of IPs belonging to local VMs is intercepted at a server, LANES installs a mapping to a special flow identifier F_{local}.

Traffic between physical servers. When LANES identifies that a packet's destination VM runs on a different server, LANES allocates any unused flow identifier F_{uw} (between the pair of communicating servers) to the pair of communicating IPs. Packets mapping to this flow identifier are rewritten (Sec. II-A) and forwarded to the physical network. LANES also configures F_{uw} at the destination server's virtual switch. Later, when packets reach the destination server, they are rewritten back to their original form and forwarded to the destination VM.

ARP queries. Tenants may use identical, conflicting IP addresses in their VMs. Left unchecked, an ARP query could get multiple answers. LANES not only contains ARP packets to their VM's virtual networks, it answers ARP requests on behalf of VMs to reduce the number of broadcast packets.

IP broadcast traffic. Broadcasts are expensive in large datacenters and can have significant negative performance impact. IP broadcast packets must also be contained to their virtual networks. Consider VM u running on server s sends a broadcast packet $P = [M_u M_\star \mid A_u A_\star]$, where $M_\star$ and $A_\star$ denote broadcast MAC and IP addresses ($A_\star$ has multiple possible values in virtual networks with multiple IP subnets). LANES forwards packet P, unmodified, to any other VM in u's virtual network that is running on server s (as for unicast traffic within a physical server), then uses two mechanisms to transmit broadcast packets between servers.

If the datacenter Ethernet network supports layer-2 multicast, LANES can be configured to create multicast groups for each virtual network. LANES generates a flow identifier $F_{u\star}$ for packet P as described for unicast packets in Sec. II-A, except LANES (i) looks up the multicast group of u's virtual network, denoted M'_u, instead of the destination server's MAC address, and (ii) installs mappings for $F_{u\star}$ on all servers that run VMs attached to v's virtual network. As an example, LANES rewrites P into $P' = [M_s M'_u \mid F_{u\star}]$. This solution requires one entry in switch MAC forwarding tables for each virtual network's multicast group.

If the datacenter does not support multicast or if Ethernet switch MAC forwarding tables cannot handle both server MAC addresses and multicast addresses simultaneously, LANES emulates broadcast using unicast packets. Again, LANES generates a flow identifier $F_{u\star}$ for packet P as described for unicast packets in Sec. II-A, except LANES (i) looks up the MAC addresses of all servers running VMs attached to u's virtual network, and (ii) installs mappings for $F_{u\star}$ on all these servers. LANES then configures servers to transmit multiple unicast packets for each broadcast packet. Emulating broadcast requires no additional entries in switch MAC forwarding tables and avoids the overhead of configuring multicast groups; this solution is preferable to multicast for virtual networks that span a small number of servers.

TABLE II: An example flow identifier table for broadcast and external packets for servers in Fig. 1.

OUTBOUND KEY				INBOUND KEY		
SRC MAC	DST MAC	SRC IP	DST IP	FLOW ID	SRC MAC	DST MAC
Broadcast from VM u:						
M_u	$M_\star$	A_u	$A_\star$	$F_{u\star}$	M_s	M'_s
External connections from VM u:						
M_u	M_χ	A_u	—	$F_{u\chi}$	M_s	M_χ
M_χ	M_u	—	A_u	$F_{\chi u}$	M_χ	M_s

Servers that receive broadcast packets from VMs in other servers use the mappings for $F_{u\star}$ to rewrite P' into P, then forward P to all local VMs attached to u's virtual network. We show an example flow identifier mapping for broadcast packets in Tab. II.

If LANES can map VMs to their IP subnets, then LANES can reduce network load by creating multicast groups for each IP subnet (at the cost of additional switch MAC forwarding table entries) or send unicast packets only to servers running VMs in that IP subnet. IP subnets could be specified in virtual network configurations or inferred from DHCP messages in virtual networks configured using DHCP.

Traffic to external networks. LANES allows VMs to communicate with the Internet or with VMs in other virtual networks by placing layer-3 routers at virtual network boundaries. These border routers serve as gateways. Packets to and from external networks are identified by having the border router's MAC address, denoted M_χ.

Consider VM u in server s sends packet $P = [M_u M_\chi \mid A_u A_\chi]$ to an external host A_χ. LANES generates an *external* flow identifier $F_{u\chi}$ for packets between VM u and A_χ. To avoid generating one flow identifier whenever a VM connects to a different external IP address, we note that LANES needs to virtualize the VM's MAC and IP addresses, but not the gateway's MAC address or the destination's IP address. LANES builds external flow identifiers $F_{u\chi}$ with 32 bits to virtualize VM addresses. External flow identifiers overwrite the source IP address of outbound packets and the destination IP address of inbound packets. LANES keeps the external IP address A_χ untouched when rewriting inbound and outbound packets. As an example, LANES rewrites packet P as $P' = [M_s M_\chi \mid F_{u\chi} A_\chi]$. Tab. II shows example external flow mappings.

LANES allows tenants with multiple virtual networks to attach VMs to more than one virtual network and route packets between their own virtual networks. As would be expected in any network, unless a VM's source IP address A_u is globally-reachable and routed to the infrastructure provider's datacenter, the tenant must provide the means for an address to be translated (using NAT) or for the packet to be tunnelled (e.g., in a VPN). In such cases, the NAT/VPN server would be part of the tenant's network and would be reachable through LANES.

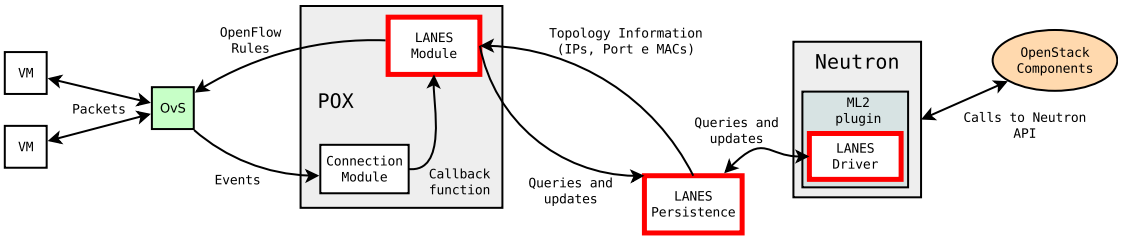


Fig. 2: Applications that compose LANES and interaction forms between them and the IaaS platform.

III. Implementation

We implemented a LANES prototype on top of OpenStack, using the POX SDN controller. OpenStack is one of the most well known open IaaS platforms; it provides applications and APIs that virtualize datacenter infrastructures, covering tasks such as server allocation, network definition, access control, firewalls, and high availability. OpenStack's *Neutron* acts as a network abstraction layer, providing APIs for tenants to express their virtual network topologies, which can then become accessible to other components.

We implemented LANES in three major modules as shown in Fig. 2. LANES's OpenStack driver implements the functions necessary to connect to Neutron and obtain virtual network topologies. LANES's network controller executes on top of the POX SDN controller and uses OpenFlow to control Open vSwitch instances running on each server in the infrastructure. LANES's persistence module keeps all topology information in a distributed database for efficient access and robustness.

Network changes such as VM instantiation or migration are sent to Neutron. Neutron, in turn, propagates changes to LANES, which can reconfigure Open vSwitch instances as necessary to guarantee correct packet delivery. When a virtualization server first boots, its Open vSwitch instance contacts the LANES controller, which configures that instance and adds it to its database. Open vSwitch instances inform the LANES network controller of any changes to its ports (e.g., link up and link down events). This allows LANES to obtain all information it needs to build flow identifiers: map VMs to virtual networks, map VMs to servers, and map servers to their MAC addresses.

IV. System Evaluation

This section describes the environment built to evaluate LANES's capabilities and performance. We show that LANES achieves forwarding performance equivalent to existing tools with negligible memory overhead while providing isolation and reducing MAC address table pressure on switches.

A. Testing environment

A physical infrastructure corresponding to part of the infrastructure of an IaaS provider was built to validate LANES's operation. Figure 3 presents the testing environment topology. It shows three virtualization servers connected to two switches. One switch is used for OpenStack control communications, to access the datacenter network, to communicate with the POX controller, and to exchange traffic with the Internet. The second switch is exclusively used for traffic between virtual

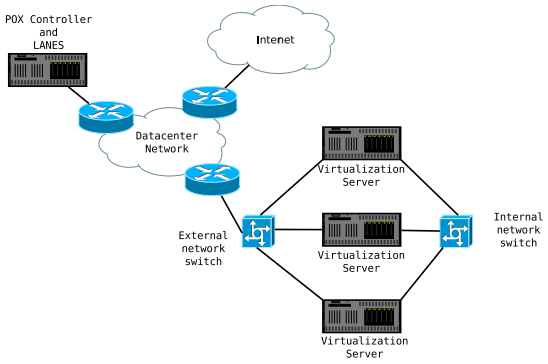


Fig. 3: Physical topology of the testing environment.

machines, corresponding to the most probable configuration of an IaaS provider. IaaS providers commonly have very big infrastructures, where servers and controllers are spread across the network. It is important to note that both switches are commodity learning Ethernet switches, without any special configuration.

We chose two switches to facilitate the understanding and verification of the correct isolation behavior between the physical and virtualized environment. The virtualization servers are three Intel Xeon E5440. The POX controller runs on an Intel Xeon E3-1240. All machines have 1 Gbps network interfaces and are connected to 1 Gbps switches. The virtual machines used in all the tests were configured with 1 processor, 1 Gbyte of RAM, 30 Gbytes of storage, and Ubuntu Linux 13.10.

On top of the physical structure, we deployed multiple virtual networks, from different tenants. Each virtual network contains multiple VMs and may span multiple servers. We considered three different software stacks for the network configuration:

1) LANES with POX module;
2) L2 switch from POX, which is offered as a reference, indicated as POX+L2;
3) OvS switch. It only forwards packets in Ethernet. It had no configured controller nor isolation between virtual networks.

In the following we evaluate isolation properties, flow establishment latency, forwarding bandwidth (in packets per second), and load overhead.

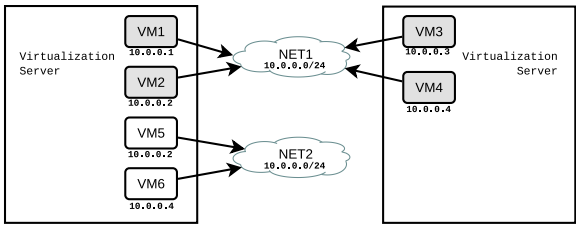


Fig. 4: Network topology during isolation tests.

TABLE III: Network isolation tests with LANES for the configuration in Fig. 4: only VMs in the same network reply to echo requests.

Sent by \ Received by	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6
VM_1	–	√	√	√		
VM_2	√	–	√	√		
VM_3	√	√	–	√		
VM_4	√	√	√	–		
VM_5					–	√
VM_6					√	–

TABLE IV: Network isolation tests with both POX+L2 and OsS stacks for the configuration in Fig. 4: all VMs in the both network reply to echo requests.

Sent by \ Received by	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6
VM_1	–	√	√	√	√	√
VM_2	√	–	√	√	√	√
VM_3	√	√	–	√	√	√
VM_4	√	√	√	–	√	√
VM_5	√	√	√	√	–	√
VM_6	√	√	√	√	√	–

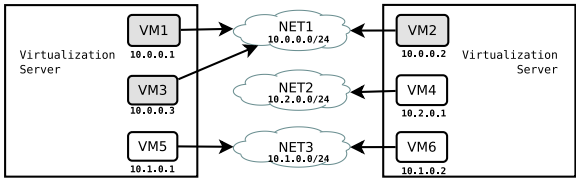


Fig. 5: Virtual network topologies used to evaluate MAC address resolution. All logical networks were implemented over the physical network in Fig. 3.

B. Isolation between virtual networks

The test for validating the isolation between virtual networks was carried out by sending ICMP packets from multiple virtual machines of different tenants using the same IP address block (10.0.0.0/24). The topology presented in Fig. 4 shows the evaluation environment, which contains a six virtual machines distributed between two tenant virtual networks. Since the networks are independent, it is expected that even using the same IP, a tenant's packet does not reach VMs in the other tenant's network.

The test consists of each VM issuing ICMP ping (echo) requests to the network broadcast address. In each column of Tab. III, we mark the virtual machines that replied to the ICMP echo request packet sent by the VM in each row. As we can see, LANES allowed the requests to reach only the machines that belong to the tenant's virtual network.

While LANES provides isolation between tenant virtual networks, POX+L2 and OvS do not. Tab. IV shows results for the same test under POX+L2 and OvS network stacks. Both mechanisms do not provide isolation and behave similarly, with all VMs from both networks receiving and replying to all ICMP echo requests.

To demonstrate the importance of isolation, we performed a throughput test. We configured a TCP flow between VM_1 and VM_3, and configured VM_6 to flood its own network with broadcast packets. That might be the behavior of a misconfigured application in its own network, or might even be an intentional malicious DoS attach on the other tenant's network. Tab. V presents the observed throughput for the tested tenant's TCP flow under each configuration. Since LANES limits the flood to that tenant's network, within one server in the infrastructure, LANES can provide better performance to the other tenant than the other two configurations.

C. Latency

The objective of latency tests is to measure the time before the start of communication between two virtual machines inside the infrastructure. During this phase many actions occur, like MAC address discovery, installation of flow rules into the OvS switches, packet rewriting (if necessary), and packet forwarding.

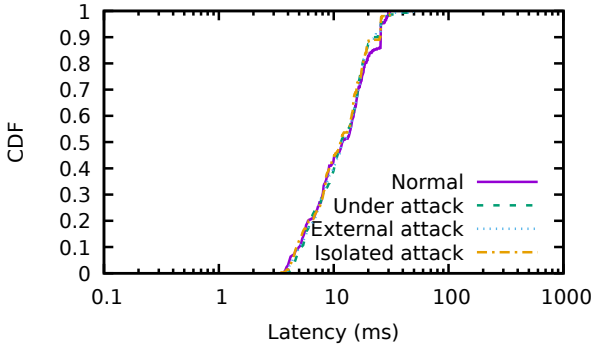
All stages prior to the beginning of communication were evaluated separately to provide a better understanding of the system's behavior. We measured the time required for MAC resolution by the ARP protocol, the time required for installing the flow rules and the time required for forwarding the packets after the flow was established. We also measured the forwarding latency of broadcast packets over unicast packets, since this type of transmission requires software processing by LANES.

MAC address resolution. Because LANES intercepts and answers all ARP packets without contacting the destination VM, this communication stage was measured separately. We measured the time interval between a VM sending an *ARP Request* and receiving an *ARP Reply*. All measurements were done capturing traffic at the VM's (virtual) network interface. The following MAC resolution tests were done using the topology presented in Fig. 5:

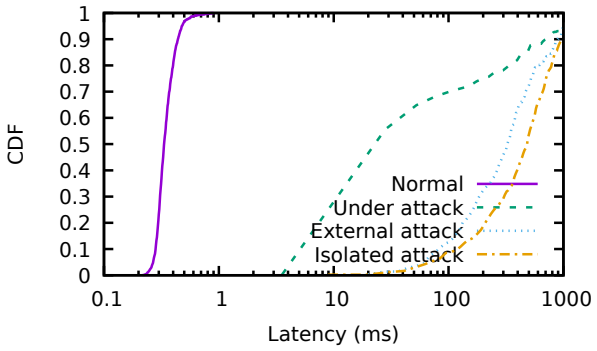
1) Normal operation: VM_1 requests the MAC address of VM_2 on the same virtual network;
2) Attack on the local network: VM_1 requests the MAC address of VM_2 while VM_3 is flooding the same local network NET_1;
3) Attack on remote network: VM_1 requests MAC address of VM_2 while VM_5 is flooding network NET_3;

TABLE V: TCP throughput between VM_1 and VM_3 in Fig. 4 while VM_6 floods its virtual network with broadcast packets.

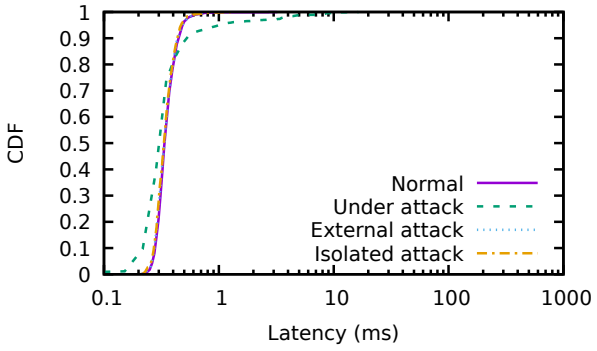
Configuration	Average Throughput	Std. Dev.
LANES	825 Mbps	2.1
POX+Switch L2	170 Mbps	5.3
OvS	295 Mbps	3.2



(a) LANES



(b) POX+L2



(c) OvS

Fig. 6: ARP resolution latency.

4) Attack on isolated network: VM_1 requests MAC address of VM_2 while VM_4 is flooding network NET_2. The difference between tests 3 and 4 is that in test 4 the attack does not generate inter-server traffic.

Figure 6 shows the results of ARP resolution using LANES, POX+L2, and OvS. Each figure contains four lines presenting results for each test.

As expected and confirmed by Fig. 6, LANES had similar response times in every type of test. This happens because LANES intercepts the ARP traffic, generates the responses internally, and prevents performance degradation due to the flooding attacks. Compared to other scenarios, response times are higher in the normal case, since the controller has to participate in all ARP processing.

In the situation where the POX controller is used with the L2 switch application, the system performance degrades

Fig. 7: Topology used to evaluate flow configuration time, packet forwarding latency, and broadcast latency.

TABLE VI: Packet forwarding latency for established flows between VM_1 and the other VMs in Fig. 7 (in ms).

Dest.	Conf.	Avg.	Max	Min	Std.
VM_2	LANES	0.35	1.19	0.26	0.078
VM_2	POX+L2	0.36	0.99	0.25	0.079
VM_2	OvS	0.37	0.95	0.27	0.080
VM_3	LANES	0.32	0.89	0.21	0.070
VM_3	POX+L2	0.32	0.74	0.21	0.063
VM_3	OvS	0.31	0.70	0.21	0.064

significantly under flood attacks, because all traffic is reaching all ports, and that must be processed by the L2 switch module. For that reason, results in Fig. 6 show that POX+L2 has the worst performance for the attacks tested.

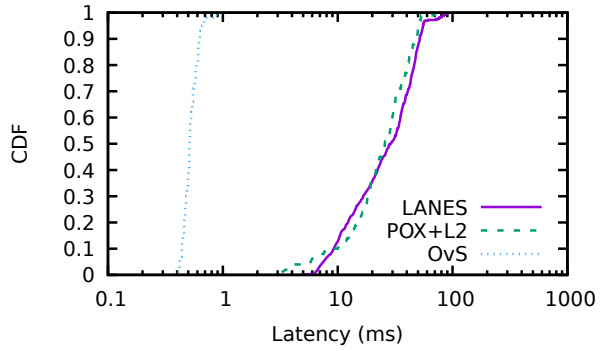
Open vSwitch has the best results in all cases. That is possible mainly because it executes within the operating system kernel and also because no type of validation is done. On the other hand, all switches and network connections are overloaded during the attacks, because it unconditionally forwards every packet that is generated during the flood.

Flow configuration latency. We also evaluate the time it takes LANES to compute flow IDs and configure flow forwading rules. As LANES rewriting rules require some computation by the controller and are installed in both the source and destination virtual switches, we expect some performance loss relative to POX+L2. We evaluate flow configuration latency on the topology shown in Fig. 7.

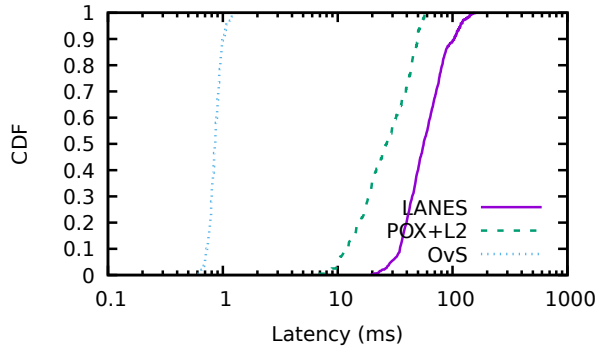
Figure 8 shows the results. In all situations, the use of a controller that installs forwarding rules reactively causes a longer delay when compared to an OvS switch without a controller. Figure 8(b) shows that configuring rules for inter-server communication has higher latency, as it requires installing rules in two OvS instances. LANES has performance similar to that of POX+L2 and we note that the latency overhead is reasonable, since flow configuration is incurred only once for each pair of communicating IP addresses (during the VMs initialization process).

Packet forwarding latency. We evaluate packet forwarding latency for established flows using the topology in Fig. 7. We measure the latency between VM_1 and VM_2 as well as VM_3.

Tab. VI shows average, maximum, minimum, and the standard deviation of packet forwarding latency for all flows and different configurations. We observe LANES forwarding overhead is minimal compared to other networking stacks, with the benefit of isolation between networks. This is relevant as forwarding overhead is incurred on each packet, while ARP resolution and flow configuration happen only once for each pair of communicating IP addresses.



(a) Same server



(b) Different servers

Fig. 8: Latency for installing OpenFlow rules on switches

Broadcast latency. We also evaluate LANES's performance for broadcast traffic emulated using unicast packets in the network topology in Fig. 7. The latency evaluation was performed sending broadcast ping packets from VM_1.

Fig. 9 shows broadcast latency for different tests. One may observe the delays added by including an SDN controller by comparing the POX+L2 and OvS lines. The reader can also observe the broadcast of LANES's software emulation of broadcast packets by comparing the LANES and OvS lines. Our current implementation in LANES emulates broadcasts using unicast packets from userspace, which achieves an average performance of $0.563\ ms$, while the SDN configuration using POX+L2 is $0.035\ ms$. We note LANES's performance could be improved by implementing the emulation of broadcast over unicast in kernel-space or by performing broadcast using multicast. As the OvS without controller does not check anything, its results were significantly better than the other two, obtaining an average response $0.0056\ ms$.

Although slower to emulate, the emulation of broadcast using unicast is worth considering in large environments, where each tenant usually has few VMs relative to the size of the datacenter. In these scenarios, emulating broadcast using unicast packets prevents the tenant from flooding the entire datacenter network. We also note that there may be only a few reasons for broadcast packets left as LANES handles ARP separately.

TABLE VII: Available bandwidth between VM_1 and the other VMs in Fig. 7 (in Gbps).

Dest.	Conf.	Avg.	Max	Min	Std.
VM_2	LANES	0.94	0.95	0.63	0.02
VM_2	POX+L2	0.93	0.94	0.74	0.02
VM_2	OvS	0.93	0.94	0.93	0.00
VM_3	LANES	0.94	1.05	0.84	0.01
VM_3	POX+L2	0.94	1.14	0.76	0.03
VM_3	OvS	0.94	0.95	0.92	0.00

D. Available bandwidth

We measured the maximum achievable TCP throughput using the same topology as in Fig. 7. We use `iperf` to generate synthetic traffic and measure bandwidth between VM_1 and the other two virtual machines, VM_2 and VM_3. The TCP connection was tested for 10 hours.

Tab. VII shows achieved throughput in each scenario. The results demonstrate that the use of an SDN controller does not influence the results and that LANES's (negligible) packet forwarding latency has negligible impact on throughput. Observe that for connections between VMs in the same virtualization server, the maximum throughput is limited only in software by OvS. Thus, the results in these cases can reach values higher than the maximum expected throughput for connections (1 Gbps).

E. Scalability evaluation

The last set of tests evaluates the system capacity, in particular, how the POX controller deals with the demand of new flow rules. The topology was composed of four virtual machines distributed over two virtualization servers and connected to the same virtual network (as in NET_1 in Fig. 4).

We configure VM_4 to generate an increasing rate of TCP connections over consecutive 120-second rounds. We show the connection generation rate and round start times in the vertical bars in the upper graph in Fig. 10. TCP flows are established with any other VM at random. LANES will then configure rules for all connections so VMs can exchange traffic. We measure the CPU utilization of the OvS process in VM_4 as it handles the highest workload and show measurements using green triangles in the upper graph of Fig. 10. The middle graph shows aggregate TCP bandwidth. The bottom graph shows ping latency, which represents flow configuration delay. The x-axis of all graphs are aligned and cover the experiment duration.

The measurement results for LANES, implemented on the POX controller, are shown in Fig. 10. They show that the developed system begins to suffer performance degradation when the rate of TCP connections is about 200 connections per second. Although it is only capable of handling well a low amount of new connections, we must remember that LANES performs several checks before deciding which flows need to be configured. The use of a production-grade controller, like ONOS, would certainly improve performance in this case.

Since LANES depends on the topology implemented in OpenStack and also needs to access the database to query it,

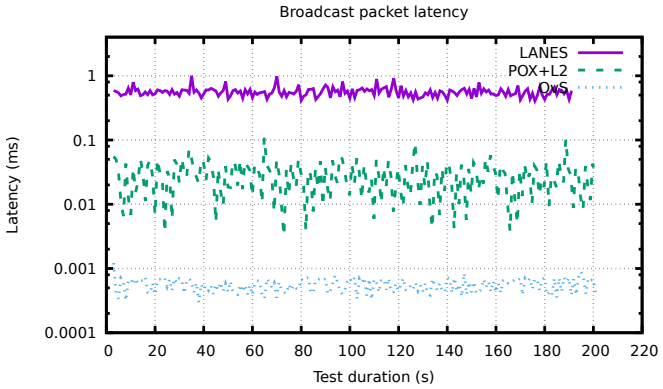


Fig. 9: Broadcast latency.

TABLE VIII: Comparison of SDN-based datacenter network virtualization solutions ('yes' is preferable).

PROPERTY	LANES	NVP	NetLord	Diverter
No encapsulation	Yes	No [12]	No [3]	Yes [13]
No header modification	No	Yes	Yes	No
IP virtualization	Yes	Yes	Yes	No
Non-IP traffic	No	Yes	Yes	No
On demand config	Yes	No	Yes	Yes
No fragmentation	Yes	Yes	No	Yes
External flows	Yes	Yes	No	Yes

LANES has an overhead higher than other network configuration stacks to handle OpenFlow messages. As LANES has not been implemented with high performance as its main objective, given POX has been implemented in Python, we believe that LANES can be improved by adjusting the used components or rewriting the functions with higher CPU usage.

V. RELATED WORK

GRE [9], and MPLS [10] require extra header fields to encapsulate packets and to provide traffic isolation. Moreover, these approaches have a limited network segment space. For instance, VLAN fields have 12 bits and allow only 4096 VLAN tags, severely limiting scalability. Another possible isolation solution through encapsulation is Q-in-Q [11], which also makes use of VLAN tags, but it uses two tags instead of one. This solution allows a much larger amount of isolated networks to run in parallel, but may still be insufficient for large datacenters. In addition, as with VLANs, the number of virtual machines in the datacenter can be so great that the switches will be unable to store all network MAC addresses in their forwarding tables. LANES does not require additional header fields and provides significantly better scalability.

NetLord [3] adopted a solution similar to ours to reduce the size of routing tables in switches inside a datacenter. NetLord encapsulates packets leaving virtual machines, creating a new packet header, addressing them to the final server in which the destination VM is located. By encapsulating the packet during transmission by the network, NetLord prevents the MAC addresses of virtual machines to be registered by switches on the way. Thus, only server addresses are stored in switch forwarding tables. Traffic encapsulation was necessary to enable one of the main features of NetLord, which is the use

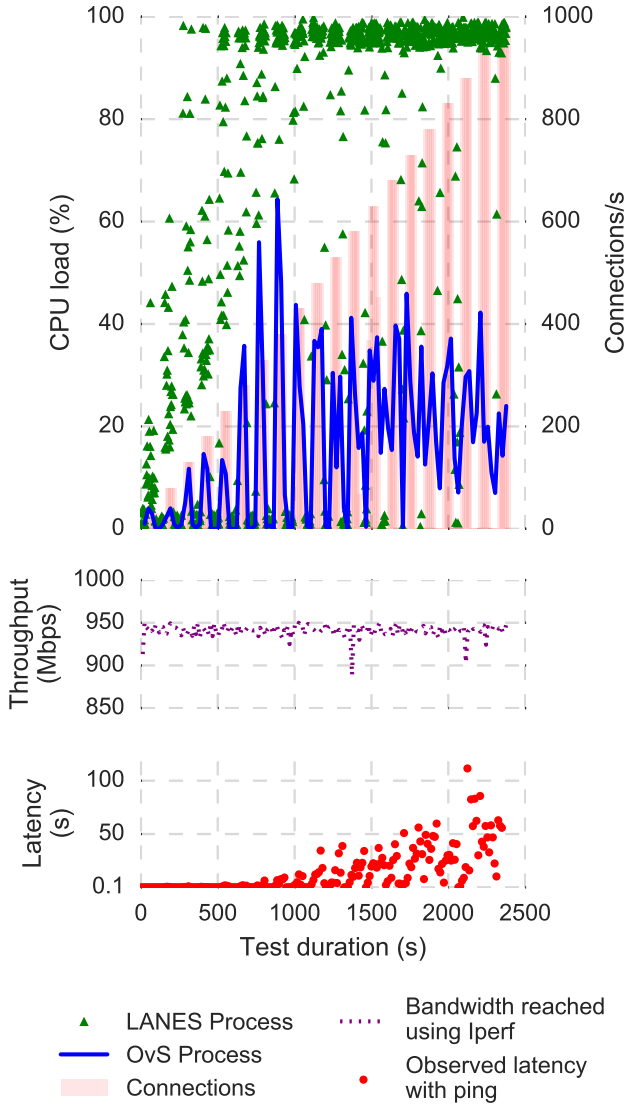


Fig. 10: Load test results for LANES.

of multiple simultaneous routes between devices. SPAIN [14] is an application by the same group that takes advantage of multiple paths to route traffic more efficiently and improve bandwidth. The main differences between LANES and NetLord is that NetLord uses encapsulation, which increases packet sizes and may cause fragmentation, and it requires changes to the virtualization layer. NetLord also does not allow direct communication between different virtual networks.

Diverter [13] is an application developed by HP Labs, which has similar objectives to LANES, such as allowing virtual networks to share the physical infrastructure while ensuring isolation. An important difference between Diverter and LANES is that Diverter assigns IP addresses to VMs; this simplifies isolation as it prevents IP address collision between different VMs. LANES allows tenants to configure arbitrary IP addresses on VMs.

Some of the authors of the initial works on SDN [15] presented NVP [12], a system to virtualize datacenter networks. The tool uses the SDN paradigm and uses network

topology information created by each customer to build a traffic isolation solution between participants. Unlike LANES, which works reactively, NVP precomputes all flow rules based on virtual network topology configurations. The controller only communicates with the switches when the topology changes, which requires change to the configured rules. Flow rule computation costs are significant, as it is necessary to cover all possible communication possibilities, ignoring the fact that VMs rarely establish all possible flows [16]. The authors report that in a network with 3,000 virtualization servers and more than 60,000 ports, flow rule computation took up to one hour to complete. In NVP, isolation between clients on the physical network is achieved using encapsulation. LANES provides on-demand configuration of flow forwarding rules and uses only packet rewriting. We summarize the trade-offs between different SDN-based datacenter network virtualization solutions in Tab. VIII.

Other previous works have proposed new traffic control and resource (bandwidth) allocation solutions for datacenter networks [17]–[20]. Those solutions allow an IaaS provider to allocate resources to tenant virtual networks and guarantee performance bounds. LANES is orthogonal to these solutions, providing the network isolation they require, and it can be combined with traffic control and resource allocation solutions to provide a richer IaaS environment.

VI. Conclusions

In this work we presented LANES, a system designed to provision and isolate virtual networks in datacenter environments. LANES rewrites packet headers to virtualize addresses, providing flexibility to hosted VMs and preventing VMs from directly impacting the physical network. LANES requires no modification to hosted VMs, puts no restrictions on VM IP addresses, does not incur encapsulation overhead, and scales to millions of VMs and thousands of servers in a single Ethernet segment. LANES also does not require advanced features and works on top of commodity Ethernet switches. LANES provides these benefits at the cost of memory utilization on infrastructure servers (hundreds of KBs), increased latencies during flow setup (hundreds of milliseconds, once per IP pair), and compute overhead to rewrite packet headers.

We integrate LANES with OpenStack, a widely used IaaS platform. LANES implements an SDN controller responsible for orchestrating the use of network resources. LANES uses Open vSwitch and standard OpenFlow rules to rewrite packets at infrastructure servers and virtualize network addresses.

We evaluated our implementation of LANES in a physical testbed. The results show that LANES effectively isolates traffic between different virtual networks. We also quantified system performance under heavy workloads. We showed that LANES can be effective in protecting the network from DoS attacks within the datacenter network. Finally, the results show that LANES achieves performance similar to that of simpler solutions after flow rules are installed.

In the future, we plan to evaluate the advantages of proactively generating rules. Although such behavior has a high flow computation cost, it has been advocated by SDN creators as preferable whenever possible.

Acknowledgments

This work was funded by FAPEMIG, CNPq, CAPES, and by projects InWeb (MCT/CNPq 573871/2008-6), MASWeb (FAPEMIG-PRONEX APQ-01400-14), and EUBra-BIGSEA (H2020-EU.2.1.1 690116, Brazil/MCTI/RNP GA-000650/04).

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *ACM SIGCOMM CCR*, vol. 39, no. 1, pp. 68–73, 2008.

[2] M. Yu, J. Rexford, X. Sun, S. Rao, and N. Feamster, "A survey of virtual lan usage in campus networks," *IEEE Comm. Mag.*, vol. 49, no. 7, pp. 98–103, 2011.

[3] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters," in *Proc. ACM SIGCOMM*, 2011.

[4] T. Sridhar, L. Kreeger, D. Dutt, C. Wright, M. Bursell, M. Mahalingam, P. Agarwal, and K. Duda, "VxLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," IETF, Tech. Rep., 2014.

[5] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the Network Forwarding Plane," in *Proc. Workshop Programmable Routers for Extensible Services of Tomorrow*, 2010.

[6] OpenStack, *OpenStack Networking Administration Guide*, OpenStack Foundation, 2013.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[8] T. Narten, M. Karir, and I. Foo, "Address resolution problems in large data center networks," RFC 6820, IETF, 2013.

[9] D. D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation (GRE)," RFC 2784, IETF, 2000.

[10] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, IETF, 2001.

[11] T. Jeffree, "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks—Amendment 4: Provider Bridges," 2006.

[12] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, "Network Virtualization in Multi-tenant Datacenters," in *Proc. USENIX NSDI*, 2014.

[13] A. Edwards, A. Fischer, and A. Lain, "Diverter: A New Approach to Networking Within Virtualized Infrastructures," in *Proc. ACM Workshop Research on Enterprise Networking*, 2009.

[14] 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.

[15] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," *ACM SIGCOMM CCR*, vol. 37, no. 4, pp. 1–12, 2007.

[16] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *Proc. ACM IMC*, 2009.

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

[18] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the Datacenter," in *Proc. ACM HotNets*, 2009.

[19] B. Heller, D. Erickson, N. McKeown, R. Griffith, I. Ganichev, S. Whyte, K. Zarifis, D. Moon, S. Shenker, and S. Stuart, "Ripcord: a modular platform for data center networking," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 457–458, 2010.

[20] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the Network in Cloud Computing," in *Proc. ACM SIGCOMM*, 2012.