Distributed Control Plane For High Performance Switch-based VXLAN Overlays

Sunay Tripathi
Pluribus Networks, Inc.
sunay@pluribusnetworks.com

Roger Chickering
Pluribus Networks, Inc.
rogerc@pluribusnetworks.com

Jonathan Gainsley
Pluribus Networks, Inc.
jon@pluribusnetworks.com

Abstract

VXLAN overlays overcome some limitations of traditional IP networks, providing mobility and scalability to virtual networks. VXLAN overlays have been implemented in software running on servers because of lack of sufficient capabilities in the previous generation of network switches. Implementing VXLAN overlays on network switches places requirements on the switch hardware that the current generation of designs is well positioned to meet. In addition to the hardware, new software is needed on the control plane of a network switch to dynamically manage VXLAN overlays. We run experiments comparing our implementation of VXLAN overlays on a network switch with an implementation of VXLAN overlays in software on a server, demonstrating that moving the VXLAN implementation onto the switch has benefits for network performance and server performance.

Keywords

VXLAN, Overlay, Performance, Virtual Network, Network Hypervisor, Distributed Network Operating System, Control Plane, Openflow, merchant silicon based switches, Openstack, Virtualization, Distributed Control Plane, Netvisor.

1. Introduction

Modern cloud architectures with multiple tenants spanning multiple data centers require virtual network architectures that are beyond the capabilities of the previous generation of networking hardware and software. To overcome the limitations of the networking hardware and software, overlay networks moved networking functionality into server hypervisors. The server initiated overlay introduces the complexity of managing the overlay network on top of and separately from the physical network and adds the overhead of encapsulating and decapsulating overlay traffic to the server.

The latest switch chips and the systems built around them have sufficient hardware capabilities to enable moving virtual networking functionality back into the network. In this paper, we look at the server overlay architecture and its advantages and disadvantages and explore making the switch operating system more programmable so we can take advantage of hardware based overlays. Moving the overlays to the switch allows us to treat the physical and virtual networks as one logical network. This also allows the servers to focus on applications and network performance without worry about network topology.

2. Server Overlays with Legacy Networks

2.1 IP Networks, VLAN, and VXLAN

A traditional IP network consists of a physical topology of pods of layer 2 switches connected to each other by routers. Each pod of layer 2 switches implements a separate layer 2 network addressed by a range of IP addresses. The routers are configured to route traffic between the layer 2 networks based on IP address.

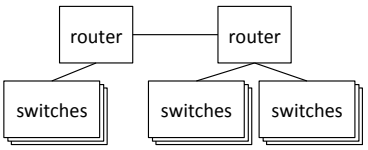

Figure 1: Traditional IP Network

VLANs [28] can be used to implement multiple virtual networks using the same physical network. A virtual network implemented with VLANs may span multiple switch pods separated by routers, but the topology of the virtual network reflects the topology of the physical network because VLAN packets are switched and routed through the network in the same way as non-VLAN packets.

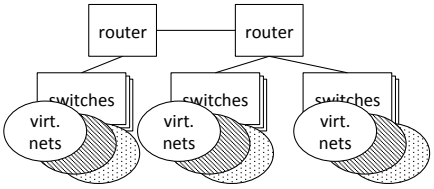

Figure 2: Virtual Networks with VLAN

Traditional IP networks and VLANs fall short of meeting the requirements of modern cloud computing for two reasons: a modern cloud deployment requires

that a virtual layer 2 network can span multiple pods of layer 2 switches separated by routers, and a modern cloud deployment may have many thousands or even hundreds of thousands of tenants, exceeding the capability of the 12 bit VLAN identifier to provide a separate virtual network for each tenant.

A virtual network using VXLAN [31] overlays addresses both problems. A packet on the virtual network is encapsulated by prepending a VXLAN header before transmission on the physical network. The VXLAN header includes a 24 bit VXLAN Network Identifier (VNI) identifying the virtual network the encapsulated packet belongs to, enabling the physical network to support over 16 million virtual networks. A single layer 2 virtual network may be spread across many switch pods separated by routers and large geographical distances.

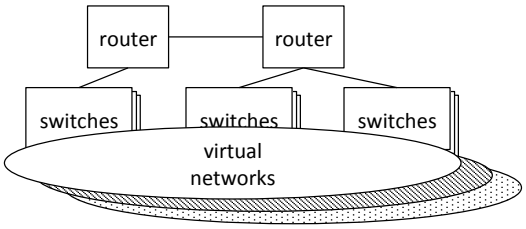

Figure 3: Virtual Networks with VXLAN Overlays

2.2 Server Implementation of VXLAN

The initial implementations of VXLAN have been in the servers that connect to the network rather than in the networking equipment. At the time VXLAN was being developed the available switches did not have VXLAN capabilities and the teams implementing VXLAN were software developers working on server hypervisors. The resulting software-only implementation has the advantage that it requires no special support from the physical network and can therefore be deployed without regard to what kind of equipment is used to implement the physical network. As long as two servers have basic IP connectivity between them, they can be configured to provide VXLAN-based virtual networks for applications and virtual machines that run on the servers.

2.3 Issues with Server Implementation of VXLAN

While implementing VXLAN overlays on servers has addressed some important shortcomings of traditional IP networks, there is potential to improve the way VXLAN overlays work by moving the implementation into the switch hardware.

Modern NICs and operating systems implement optimizations for TCP/IP over Ethernet, including virtual switching offload [29], checksum offload for TCP and UDP, large segment receive, and transmit offload [30]. Together, these features provide tremendous performance and scaling benefits by offering stateless offload. Server based overlays are software constructs that don't benefit from these optimizations. Combined with the additional overhead of encapsulating and decapsulating VXLAN packets in software, the server uses significant CPU resources handling network traffic (see section 6) that otherwise would be available for application use. The server may not be able to encapsulate and decapsulate the traffic in software at a high enough rate to utilize the network capacity available, so that the encapsulation and decapsulation becomes a bottleneck for network traffic.

The discovery phase of some implementations of VXLAN overlays uses multicast, which is unreliable and not implemented well on some switches. In some cases this may make the VXLAN overlay dependent on the underlying network implementation.

The security and monitoring tools that administrators use to monitor their networks must be updated in order to effectively operate in the presence of overlays. When encapsulation and decapsulation are performed on the server, all traffic through the network is encapsulated, reducing the effectiveness of traditional network monitoring tools.

3. Hardware Support for Switch Implementation of VXLAN Overlay

The alternative to a server implementation of VXLAN overlay is to implement VXLAN overlay in the network switch the server is connected to. For this to be practical, the network switch must meet the requirements laid out in this section.

The authors have worked with a team to implement Netvisor, a distributed network operating system. We have used Netvisor to prototype the ideas presented in this paper. Our experience running Netvisor with the current generation of switch chips from Broadcom and Intel and a range of CPU chips from Intel have shown that modern merchant silicon hardware meets the requirements in this section.

3.1 Software Programmability

Virtual networks supporting multiple applications and tenants with VXLAN overlays require different treatment based on traffic characteristics, security needs, latency and bandwidth requirements. The network must understand virtual machine creation and migration and honor policies configured by the network operator. Building this logic into a switch

ASIC would make the switch ASIC complex and inflexible.

In order to provision VXLAN overlays on demand, the software control plane controls all MAC learning on the switch. This means the switch chip must provide a feature for disabling MAC learning and sending packets that don't have MAC table entries to the CPU instead of flooding them. Software running on the control plane CPU handles MAC table misses as described in section 5.2.

Managing ARP traffic by the control plane is an important aspect of VXLAN overlay implementation as described in section 5. The switch chip must be configurable to send all ARP packets to the control plane CPU rather than switching the ARP packets.

The switch hardware must allow the control plane CPU to manage the entries in the hardware MAC table, adding and removing entries as needed to support virtual networks. The switch hardware also needs to enable the control plane CPU to configure encapsulation and decapsulation rules.

3.2 High Bandwidth Interconnect

To support dynamic virtual networks the switch chip may send hundreds of thousands of packets per second to the CPU control plane for MAC table misses and ARP traffic. In response to those packets the CPU control plane may make hundreds of thousands of configuration changes to the chip hardware per second and may inject hundreds of thousands of packets into the switch per second. The interconnect between the switch chip and the CPU must have enough bandwidth to support these packets and configuration changes.

The 2-4 lanes of PCI-Express Generation 2 connecting a modern switch chip to a modern CPU enables up to 16Gbps of bandwidth between the CPU and the switch chip. This is more than adequate for handling the packets and configuration changes needed to implement VXLAN overlays using the switch chip hardware. DMA is used to transfer packets between the CPU and the switch chip.

3.3 Control Plane CPU and Memory

In order for the control plane CPU to handle the MAC table misses and ARP traffic from the switch and configure the switch with dynamic VXLAN overlays, the control plane CPU hardware needs a powerful platform with multiple CPU cores and enough memory to store large tables for managing the endpoints in all the virtual networks [3][4][5][6]. Our experience with Netvisor has taught us that modern Atom processors from Intel with at least 4GB of RAM can support virtualized networks with hundreds of virtual machines. To scale to millions of virtual machines requires Xeon processors and at least 64GB of RAM.

4. Software Support for Switch Implementation of VXLAN Overlay

This section describes some Netvisor features that we used to implement our prototype of VXLAN overlay using switch hardware. Netvisor is a distributed switch operating system based on an open source kernel.

4.1 Software Access to Switch Hardware

The switch chip is integrated into the server operating system over PCI-Express. The switch register space is memory mapped into software. Software manages the MAC, IP and flow tables. The design for control plane I/O is fully multithreaded and similar to Crossbow Virtualization Lanes with a dynamic polling[4][21] architecture. There is no hardware MAC learning on the switch chip and when there is a MAC table miss in hardware the packet is forwarded to software. As shown in Table 1, software keeps a large MAC table in main memory and uses the hardware table as a cache that is updated on a miss. The access into the switch for table updates is fully multithreaded and is protected by fine-grained locks to provide high bandwidth, low latency access. MAC learning is a compute intensive operation since ARP packets are selectively sent to individual ports as described in section 5 and is dependent on the performance of the control processor.

Table 1. Software MAC Table

Chip	Hardware MAC Table	Software Learning Rate	Software MAC Table
Alta/ Trident2	64k-288k	15k/sec	1.2M

4.2 Fabric

A fabric is a collection of switches that share configuration and state information. Switches in a fabric work together to provision the resources allocated by the configuration and to manage state information across the fabric.

A switch discovers other directly connected switches using layer 2 and layer 3 discovery protocols. Switches separated by IP routers may be configured into a single fabric by an administrator. The fabric synchronizes the configuration and state across all switches in the fabric using TCP/IP.

4.3 Fabric Protocol

Configuration and state is shared between nodes using a multi-threaded event queue[27]. Strict synchronization for configuration changes is achieved

using a three-phase commit protocol[19][20] to ensure consistency across all switches.

State changes are used to inform the fabric of the state of each switch. Each switch has its own state, so the three-phase commit protocol is not needed for state changes. Instead, each switch informs the other switches in the fabric whenever its state changes. Each switch maintains tables in memory to represent the state of the other switches in the fabric.

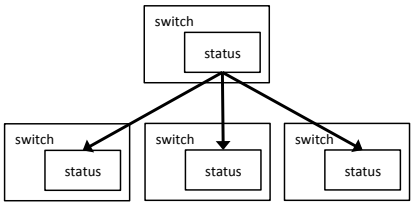

Figure 4: State Changes

A switch can make a remote procedure call to another switch. Remote procedure calls are non-transactional and don't explicitly imply any update to state tables. A remote procedure call may have arguments and return values.

On each switch a separate message queue is maintained for every other switch. Switch-to-switch communication happens in parallel so that adding more switches to the fabric doesn't slow down configuration changes or status updates.

The fabric protocol uses TCP/IP for configuration, state change, and remote procedure call messages. The switches that make up a fabric can be separated by other switches, routers, or tunnels. As long as the switches have IP connectivity with each other they can share fabric state and configuration.

4.4 Fabric-wide vport Table

In Netvisor a vport is a data structure that represents an endpoint attached to the network. vports are maintained for servers, virtual machines, switches, and any other device connected to a switch port. Each vport contains the switch and physical port the vport is associated with along with the MAC/IP/VXLAN information for the virtual or physical machine. The vport also contains metadata related to state, policy and usage. vports are stored in shared state tables. Each switch in the fabric stores vports for all the switches in 3the fabric, and state changes to vports are propagated through the fabric using the fabric protocol. The vport table has indexes based on both MAC address and IP address. The switch chip's hardware MAC table caches the vports that are active on each switch.

The tunnel-id field of a vport is only of interest to the local switch and therefore is not propagated through the fabric with the rest of the vport.

4.5 Tunnel and Flow Tables

The interface to the switch chip for VXLAN overlay involves creating tunnels and flows. Tunnels are for encapsulating and decapsulating VXLAN packets, and flows are for (among other things) directing traffic for specific destination MACs into tunnels. Software maintains a tunnel table and a flow table to track what has been configured in the hardware.

4.6 Orchestration

An orchestration system is responsible for mapping a virtual network to a VLAN when placing a virtual machine on a server. Netvisor provides a rich set of APIs that applications can use to query and manage Netvisor's configuration and state. For OpenStack [25] deployments a plugin that runs on the server uses the Netvisor APIs to look the virtual network up in the switch's configuration to determine which VLAN to use.

5. Switch Implementation of VXLAN Overlay

In this section we describe how the programmable switch chip and the fabric-wide vport table are used to implement virtual networks using VXLAN overlays.

A virtual network is configured by the administrator who supplies the name and the VXLAN VNI when creating a vnet. The fabric synchronizes the vnet across all the switches in the fabric. Once the vnet is configured, the switches in the fabric dynamically create switch-to-switch tunnels as needed to enable endpoints to communicate on virtual networks across switches. The following subsections describe the mechanics of dynamic provisioning of VXLAN overlays.

5.1 Sample Network

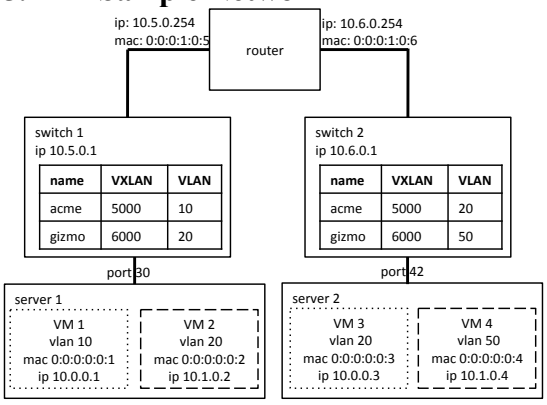

Figure 5: Sample Network

The sample network consists of two switches, two virtual networks, two servers, and four virtual machines. The switches are on different IP networks and are separated by a router.

5.2 Software MAC Learning

The switch chip is configured to not automatically learn MAC addresses in hardware. The hardware tables are a cache of the vport table with the least recently used entry replaced on a hardware miss.

When the switch chip receives a packet from the network whose source and/or destination mac address is not in the hardware table, the switch chip sends the packet to the CPU rather than switching the packet. In addition, the switch chip is configured to send all ARP packets (including ARP packets encapsulated in VXLAN headers) to the CPU rather than switching them.

Upon receipt of an ARP request, a source mac miss packet, or a destination mac miss packet, the CPU control plane on the switch that receives the request uses the vnet table to map the VLAN of the packet to its VXLAN and looks up the source MAC and VXLAN of the ARP request in the fabric-wide vport table. If an entry exists and the packet is an IP packet and the sender IP address in the ARP request is different from the IP address stored in the vport, the vport and the IP address index are updated. If no entry exists, a new one is created. A hardware MAC table entry is added for the vport if it is not already active. Then the vport is propagated to other switches in the fabric.

In the sample network, suppose that an application on VM 1 initiates a TCP connection to a server running on VM 3. At this point the vport table for the fabric contains no entries. The network stack on VM 1 sends an ARP request to find the MAC address of the endpoint with VM 3's IP address 10.0.0.3.

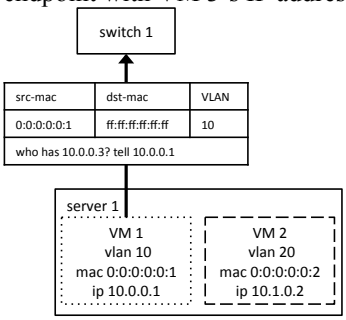

Figure 6: ARP Request

The switch chip is configured to send all ARP packets to the CPU control plane. Upon receipt of the ARP request the CPU control plane on switch 1 adds a vport entry for VM 1 and configures the MAC table on the switch chip.

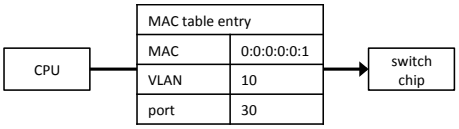

Figure 7: Configure MAC Table Entry

Then the CPU control plane on switch 1 propagates the vport to the other switches in the fabric, switch 2 in this case.

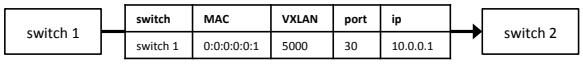

Figure 8: Propagate vport

5.3 Endpoint Discovery

At this point, the control plane CPU on switch 1 has created a new vport entry based on the ARP request it received. Now switch 1 handles the ARP request. The control plane CPU looks up the target IP and VXLAN in the fabric wide vport table to see if an entry exists for the target of the ARP request. If no entry exists the control plane CPU injects the ARP request into the switch chip, instructing the chip to send the ARP to non-switch ports other than the one on which the ARP request was received.

In addition, the CPU removes the VLAN tag from the ARP request and then forwards the ARP request to each switch in the fabric encapsulated in a VXLAN header. Note that the CPU control plane can send a VXLAN encapsulated packet to another switch without first setting up a tunnel in the switch chip hardware because the CPU control plane lays out the VXLAN encapsulated packet in software before sending it. A tunnel (see below) is needed to have the switch chip perform the encapsulation.

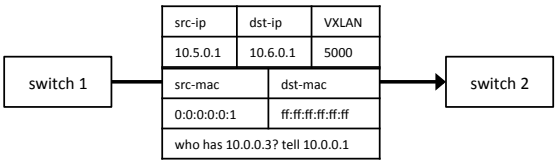

Figure 9: Forward ARP Request

The switch chip on switch 2 receives the encapsulated ARP request and forwards it to the CPU control plane. The CPU control plane looks up the VXLAN in the vnet table to find the local VLAN for the virtual network, removes the VXLAN header, tags the packet with the VLAN and sends the packet to the switch chip to be sent to all ports that are not connected to other switches.

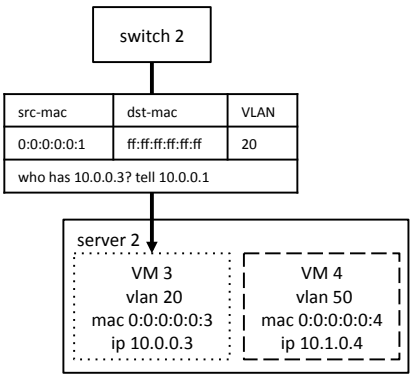

Figure 10: ARP Request Sent from switch 2

Since VM 3 is the owner of the IP address 10.0.0.3, VM 3 sends an ARP response.

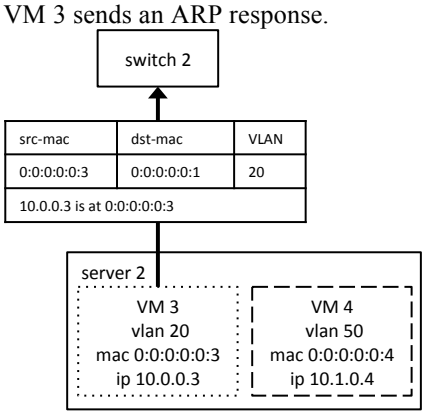

Figure 11: ARP Response

The switch chip on switch 2 sends the ARP response to the CPU control plane. The CPU control plane uses the VLAN on the ARP response to find the corresponding VXLAN VNI in the vnet table. The CPU control plane creates a vport entry for VM 3, configures the switch chip with a MAC table entry corresponding to the vport, and propagates the vport entry to the other switches in the fabric, in this case switch 1. The fabric-wide vport table now looks like:

switch	MAC	VXLAN	port	ip
switch 1	0:0:0:0:0:1	5000	30	10.0.0.1
switch 2	0:0:0:0:0:3	5000	42	10.0.0.3

Switch 2 uses the destination MAC address and the VXLAN VNI to look up the vport for the destination of the ARP response. Switch 2 finds the vport that shows switch 1 as the owner switch.

The fact that VM 3 has sent an ARP response to VM 1 indicates that VM 1 and VM 3 are about to engage in IP traffic. At this point switch 2 knows that VM 1 is connected to switch 1 and VM 3 is connected to port 42 on switch 2. So the CPU control plane on switch 2 configures its switch chip for tunneling packets from VM 3 to switch 1.

The first step is to consult the software tunnel table to see if there is a tunnel set up from switch 2 to switch 1. Since no overlay traffic has been transmitted from switch 2 to switch 1 yet there is no tunnel. So the CPU control plane configures the switch chip with a new tunnel from switch 2 to switch 1.

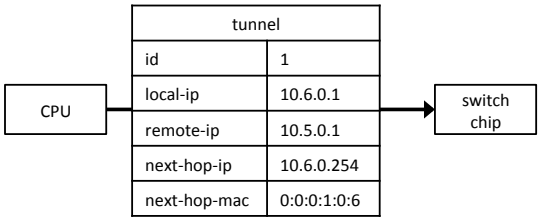

Figure 12: Configure Tunnel in Switch Chip

The CPU control plane also configures the switch chip with a flow for decapsulating packets that arrive on the tunnel with VXLAN VNI 5000. Arriving packets with VXLAN VNI 5000 are decapsulated by the switch chip and tagged with VLAN 20 and then switched. The flow handles all traffic arriving through the tunnel with VXLAN VNI 5000 and so only has to be installed the first time the CPU control plane on switch 2 determines that traffic on VXLAN VNI 5000 will be exchanged with switch 1.

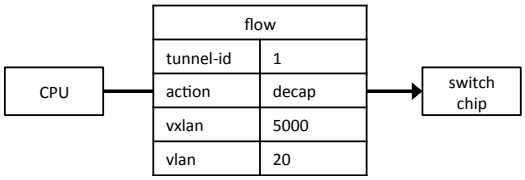

Figure 13: Configure Decap Flow in Switch Chip

Now that the tunnel is set up, the CPU control plane on switch 2 looks at the vport entry for VM 1 and notes that the tunnel-id field is not set. Recall that the tunnel-id field is not propagated to other nodes in the fabric and is used to keep track of the tunnel for encapsulating packets sent to non-owner switches.

The CPU control plane on switch 2 updates the tunnel-id field of the vport with the ID of the tunnel to switch 1 and installs a flow into the switch chip telling the hardware to send packets destined for VM 1 into VXLAN tunnel 1:

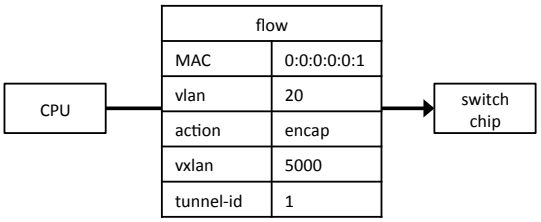

Figure 14: Configure Encap Flow in Switch Chip

Now the CPU control plane on switch 2 removes the VLAN tag from the ARP response, prepends a

VXLAN header, and sends the ARP response to switch 1.

The switch chip on switch 1 sends the encapsulated ARP response to the CPU control plane. The CPU control plane uses the VXLAN VNI and the inner source MAC address to look up the vport for the source of the ARP response, retrieving the vport entry for VM 3. The CPU control plane uses the VXLAN VNI and the inner destination MAC address to look up the vport for the destination of the ARP response. Switch 1 is the owner of the vport and the endpoint is attached to port 30. The CPU control plane strips off the VXLAN header, tags the ARP response packet with the VLAN, and sends the packet out port 30 by injecting the packet into the switch chip.

At this point the vport tables are configured with the endpoint information necessary for provisioning connectivity between VM 1 and VM 3. A VXLAN tunnel is configured in hardware for sending traffic from VM 3 to VM 1. However there is as yet no tunnel in hardware for traffic from VM 1 to VM 3.

The next packet that VM 1 sends to VM 3 is sent by the switch to the CPU control plane because there is still no hardware mac table entry for VM 3's mac address. The CPU control plane looks up VM 3's mac address in the vport table, sees that VM 3 is connected to switch 2, and configures a hardware VXLAN tunnel, decap flow, and encap flow for tunneling packets between VM 1 and VM 3.

The example packets in the last two sections were ARP packets because ARP packets are typically the first packets sent by a host when connecting to another host. If hosts are configured with static ARP entries then provisioning of vports and tunnels works the same way, except that instead of ARP packets the switch sends source and destination mac miss packets to the CPU control plane. The CPU control plane creates vport entries and sends VXLAN encapsulated packets to other switches for destination mac miss packets in the same way that the CPU control plane handles ARP packets.

5.4 ARP Optimization

Once an endpoint has a vport entry, ARP optimization is used for lighter weight provisioning of hardware MAC entries and VXLAN tunnels.

Suppose server 2 starts another virtual machine on the acme virtual network, and software on the virtual machine initiates a TCP connection to VM 1. The virtual machine sends an ARP request:

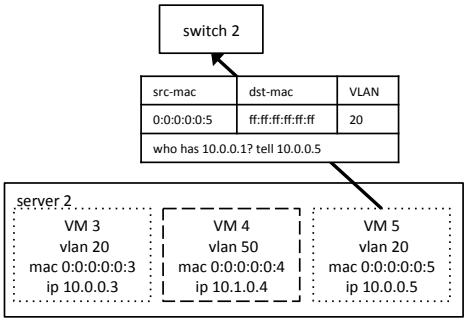

Figure 15: ARP request

As before the switch chip sends the ARP request to the CPU control plane, and the CPU control plane configures a new vport and MAC table entry for the VM 5. The CPU control plane propagates the new vport to the other switches in the fabric.

However this time it is not necessary to send the ARP request to other switches in the fabric. Switch 2 already has a vport for VM 1, which contains everything switch 2 needs to provide an ARP response. Switch 2 creates the ARP response and sends it to VM 5.

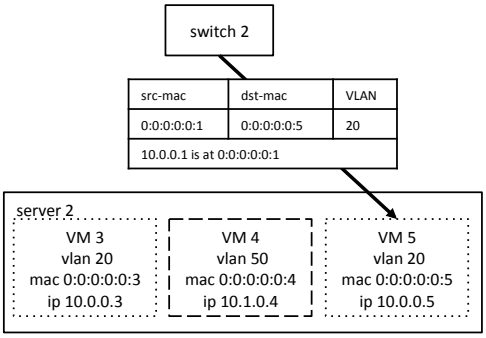

Figure 16: ARP response

Note that the ARP response comes from the CPU control plane on switch 2, not from VM 1. VM 1 never receives the ARP request from VM 5 because the CPU control plane on switch 2 handles the ARP request directly without forwarding it anywhere else..

5.5 Switch-to-Switch Tunnel Scaling

Each switch in the fabric may need a tunnel to every other switch in the fabric for overlay traffic. In a fabric with N switches, each switch has up to N – 1 tunnels.

Decapsulation flows are configured as needed. A decapsulation flow is needed for each active VXLAN and switch combination. In a fabric with N switches and M VXLAN-based virtual networks, each switch may need as many as (N – 1) * M decapsulation flows. In practice far fewer decapsulation flows are needed because not all VXLANs are active between all pairs of switches.

Each switch needs an encapsulation flow for each endpoint on another switch that a local endpoint communicates with using VLXAN overlay.

The current generation of switch chips can store 24K to 56K flows in hardware. This is the working set of decapsulation and encapsulation flows that are used to implement VLXAN overlay in hardware. In most cases the majority of flows will be encapsulation flows, meaning the current generation of switch chips can handle simultaneous traffic between local endpoints to between 24K and 56K endpoints on other switches reachable through overlays.

Any traffic that exceeds the limit of the number of flows results in hardware MAC table misses, causing the switch chip to send packets to the CPU control plane.When the CPU control plane receives a packet due to MAC table miss, the CPU control plane chooses the least recently used flow or flows to remove from the switch chip so that new flows can be added that enable the packet to be handled by hardware.

As long as the majority of traffic at a given time can be handled by the switch chip in hardware, VXLAN overlay traffic is handled at line rate.

6. Performance

To quantify the performance benefit of switch overlays over server overlays, we ran a series of experiments with two Linux servers connected to two switches separated by a router. The measurements were done using the "uperf" tool that allows us to control the packet size and number of connections. Previous efforts[35] to measure tunnel performance used the netperf TCP test that uses 1500 byte or larger packet sizes and does not show the impact when smaller size packets are used. The servers were running Ubuntu 3.13 (which was the most stable for server overlays) and using Intel E3-1270 V2 CPUs (8 cores at 3.5Ghz) with 16GB memory and Intel Niantic 10Gbps NICs. The VMs were created with 8 cores assigned to them and the experiments were never CPU limited. The CPU utilization was measured by looking at /proc/stats and normalizing for 10Gbps bandwidth.

All the experiments used 10 threads (connections) between client and server where the client sent 64 byte, 512 byte, and 1450 byte packets in TCP streams (32K window size). The maximum MTU was kept at 1450 bytes to ensure that there was no fragmentation in the tunnel case.

Baseline

The first set of experiments establishes a baseline for how fast the two servers can communicate over the test network. We configured the servers to use the router as their default gateways and ran traffic between them. We also ran a baseline using a VNIC configured over OVS (Open vSwitch [12]) and a set of virtual machines configured to use the VNICs but without any VXLAN. In all three cases, the traffic was routed via the router.

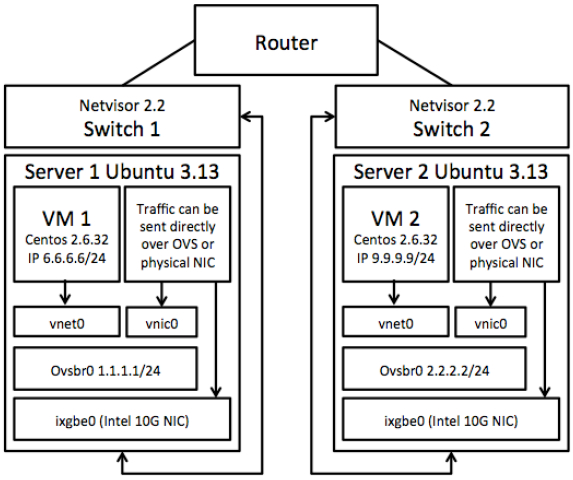

Figure 17: Configuration to measure Base Performance

The table below shows the native server-to-server performance directly over Intel 10Gbps NIC (ixgbe driver) and performance using a VNIC over OVS. There are no virtual machines or tunnels involved in baseline performance. As can be seen from the performance data, the VNIC over Linux OVS itself adds some performance penalty for small packet sizes. The bigger packet sizes got the same performance.

Baseline to measure non-VXLAN performance			
Scenario	64B packets	512B packets	1450B packets
Server-to-Server over ixgbe	7.0Gbps	9.34Gbps	9.35Gbps
Server-to-Server VNIC over OVS	6.33Gbps	9.34Gbps	9.35Gbps
VM to VM	4.45Gbps	8.88Gbps	9.24Gbps

Server to Server Overlay

For the second set of experiments we configured OVS to establish a tunnel between the two servers. A VNIC over OVS was created to send packets over the VXLAN tunnel from the server itself. Then virtual

machines were created on both servers with IP addresses on the same subnet so the virtual machines could communicate over the VXLAN tunnel.

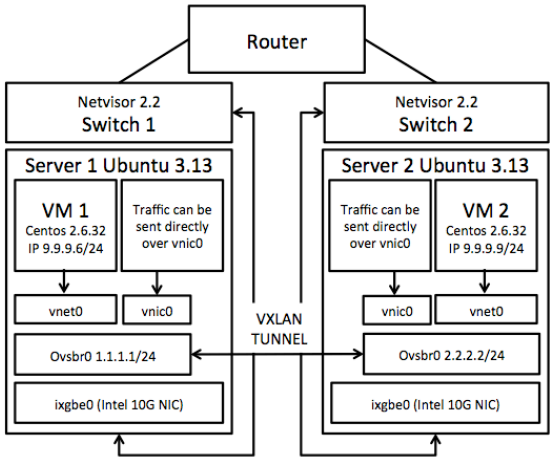

Figure 18: Configuration to measure Server Overlay Performance

The table shows performance drops significantly compared to the non-VXLAN case.

Performance when using Server based VXLAN Overlays			
Scenario	64B packets	512B packets	145B0 packets
Server to Server over OVS (Bandwidth)	4.95Gbps	6.36Gbps	6.32Gbps
Server to Server over OVS receiver CPU utilization (normalized to 10Gbps)	77%	30%	19%
VM to VM	3.46Gbps	5.31Gbps	5.56Gbps

Switch to Switch Overlay

For the final experiment we removed the VXLAN tunnel from the servers and instead configured a VXLAN tunnel between the switches.

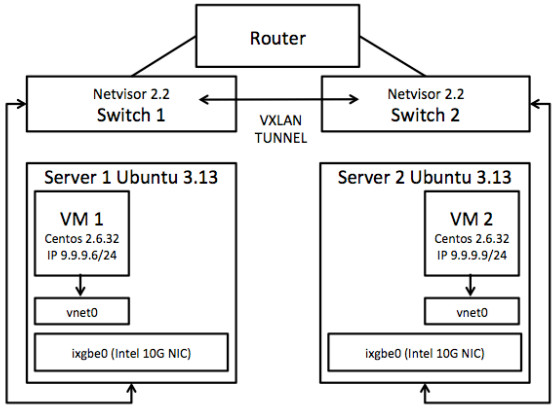

Figure 19: Configuration to measure Switch based Overlay Performance

The table shows that there is very little performance loss when a VXLAN tunnel is configured between the switches.

Performance when using Switch based VXLAN overlays			
Scenario	64B packets	512B packets	1450B packets
Server to Server over OVS (bandwidth)	6.09 Gbps	9.0Gbps	9.0Gbps
Server to Server over OVS – receiver cpu utilization (normalized to 10Gbps)	60%	18%	17%
VM to VM	4.35Gbps	8.71Gbps	8.93Gbps

6.1 Server Overlay vs Switch Overlay Performance and CPU utilization Comparison

Figure 20 compares server-to-server throughput without overlay, server based overlay, and switch based overlay. As seen from the graph, the switch based overlay, which adds no CPU overhead to the servers, achieves the same performance as the non overlay experiment while the server based overlay has a high performance penalty. The penalty is especially severe in case of small packet size (64 bytes). Koponen[35] showed similar findings with GRE based tunnels where CPU utilization more then doubled and throughput dropped from 9.3Gbps to 2.4Gbps. Our

results show that server VXLAN based overlays have better performance than GRE tunnels but lack of hardware offload (Section 2.3) with tunnels hurts performance and increases CPU utilization on servers.

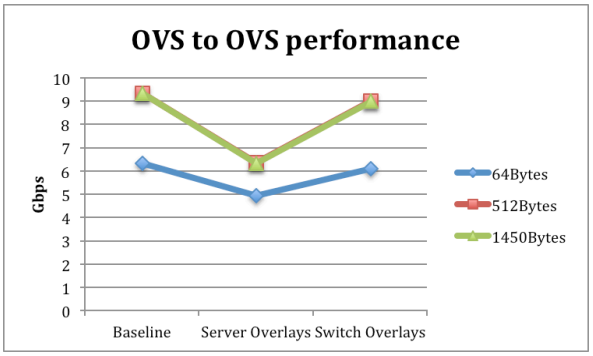

Figure 20: OVS to OVS performance for different packet sizes

Figure 21 shows that in addition to lower achieved bandwidth, CPU utilization was higher with server overlay. The CPU utilization was measured as a percentage of total CPU available and normalized for 10 Gbps of throughput.

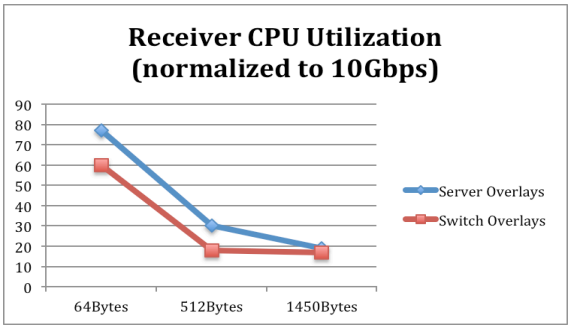

Figure 21: CPU utilization at Receiver

On the receiver side, there were a few cores that were getting close to 100% utilization while others were fully idle. Most modern kernels do the bulk of the receive side processing using kernel threads which have CPU affinity. The CPU percentage is the part used of total available CPU.

On the sender side, the CPU utilization varied heavily so meaningful conclusions couldn't be drawn. We were not able to investigate this fully due to lack of time but it appears that the following factors contributed to the high variance:

- Low CPU utilization combined with high I/O wait time
- Bulk of send side processing happening in user context (due to lack of H/W offload with tunnels)
- Heavier thread migration between CPUs

Perhaps increasing the TCP window size would have streamlined some of these issues but this exercise is left for future work.

Figure 22 shows the bandwidth achieved when sending different sized packet streams from a VM to another VM. In the baseline case, the VM had IP addresses on their respective local subnets and the traffic was routed to them. The next set of measurement involved the VMs having IP addresses on the same subnet and a VXLAN tunnel was created between servers and subsequently on the switches.

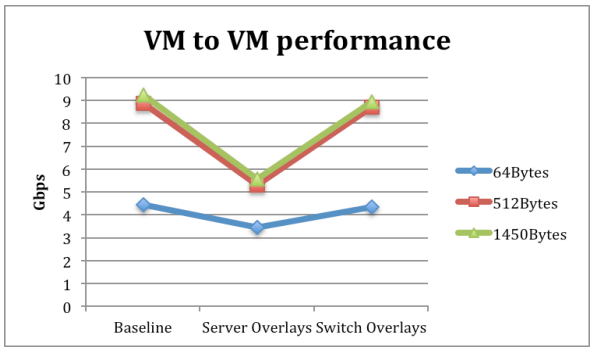

Figure 22: VM to VM Performance for different packet sizes

As evident from the graph, the server overlays added significant performance penalty for throughput.

7. Related and Future Work

vmware NSX [33] includes a commercial implementation of server overlay networks.

Onix [7] is a distributed networking control plane.

Future work includes integrating with OpenDaylight [34] and continuing our work with OpenStack [25] to enable provisioning of switch-based overlays using OpenDaylight and OpenStack..

More performance measurements at scale need to be done that include multiple virtual machines on servers and multiple servers in the networks. The CPU utilization measurement is often not very accurate on an unloaded machine due to thread migrations and unnecessary context switches but a fully loaded machine where only throughput is measured gives a better picture of resource utilization.

8. Conclusion

Virtual networks implemented using VXLAN overlays offer important advantages over virtual networks implemented using VLAN: a single physical network can support more virtual networks, and the topology of

the virtual network can be independent of the physical network.

VXLAN overlays may be implemented entirely in servers, using the physical network for tunneling virtual network traffic between servers.

As we have shown in this paper, an alternative with significant performance advantages is to implement VXLAN overlays using modern switch hardware in conjunction with a distributed CPU control plane.

The work presented in this paper was a joint effort between the authors and the rest of the Pluribus Networks engineering team. The authors acknowledge and thank the entire team without which switch based VXLAN overlays would not have been possible.

9. References

[1] Ozdag Recep. Intel Ethernet Switch FM6000. DOI=http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf.

[2] Broadcom. Strata XGS Trident II Ethernet Switch Series. DOI=https://www.broadcom.com/products/Switching/Data-Center/BCM56850-Series

[3] G. Liao, D. Guo, L. Bhuyan, S. King. 2008. Software techniques to improve virtualized I/O performance on multi-core systems. 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 161-170.

[4] S. Tripathi, N. Droux, T. Srinivasan, K. Belgaied. Crossbow: From H/W virtualized NICs to virtualized networks. Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures. VISA 2009, 53-62.

[5] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. 2009. Portland: A scalable fault-tolerant layer 2 data center network fabric. ACM Sigcomm 2009 conference on Data Communications, 2009.

[6] A. Greenberg, J. Hamilton, D.A.Maltz, and P.Patel. 2009. The cost of the cloud: research problems in datacenter networks. SIGCOMM Computer Communication Review. ACM, 68-73.

[7] T.Koponen, M.Casado, N.Gude, J.Stribling, L.Poutievski, M.Zhu, R.Ramanathan, Y.Iwata, H.Inoue, T.Hama, and S.Shenker. 2010. Onix: A distributed control platform for large scale production networks. In USENIX OSDI, 2010.

[8] N.McKeown, T.Anderson, H.Balakrishnan, G.Parulkar, L.Peterson, J.Rexford, S.Shenker, and J.Turner. 2008. Openflow: enabling innovation in campus networks. ACM Sigcomm CCR. 2008.

[9] Open Computer Project. DOI=http://www.opencompute.org/wiki/Networking/SpecsAndDesigns

[10] A. Myers, E. Ng, and H. Zhang. 2004. Rethinking the service model: scaling Ethernet to a million nodes. HotNets, November 2004

[11] Kim, Changhoon, Matthew Caesar, and Jennifer Rexford. 2008. Floodless in seattle: a scalable ethernet architecture for large enterprises. *ACM SIGCOMM Computer Communication Review*. Vol. 38. No. 4. ACM, 2008.

[12] E. Keller and J. Rexford. 2010. The ”platform as a service” model for networking. In Proceedings of the 2010 internet network management conference on Research on enterprise networking, INM/WREN’10, Berkeley, CA, USA, 2010. USENIX Association.

[13] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Sharing the data center network. 2011. In Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI). USENIX Association.

[14] R. Cheveresan, M. Ramsay, C. Feucht, and I. Sharapov. Characteristics of Workloads used in High Performance and Technical Computing. In International Conference on Supercomputing, 2007.

[15] S. Brin and L. Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems, 30(1-7), 1998.

[16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. USENIX Symposium on Operating Systems Design and Implementation, 2004.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. ACM SIGOPS Operating Systems Review, 37(5), 2003.

[18] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In USENIX Conference on File and Storage Technologies, 2002.

[19] Dale Skeen. Non blocking commit protocols. 1981. Proceedings of the 1981 ACM SIGMOD international conference on Management of data. Pages 133-142. ACM New York, NY, USA 1981.

[20] Idit Keidar and Danny Dolev. 1998. Increasing the Resilience of Distributed and Replicated Database Systems. In the Journal of Computer and System Sciences (JCSS) special issue with selected papers from PODS 1995, 57(3) pages 309-324. December 1998.

[21] Sunay Tripathi , Nicolas Droux , Kais Belgaied , Shrikrishna Khare. 2009. Crossbow Virtual Wire: Network in a Box. LISA '09, USENIX. 2009.

[22] J. Bonwick, M. Ahrens, V.Henson, M.Maybee, and M. Shellenbaum. 2003. The Zettabyte file system. Technical Report, Sun Microsystems. 2003.

[23] Uri Lublin and Yaniv Kamay and Dor Laor and Anthony Liguori. 2007. Kvm: the Linux Virtual Machine Monitor. Proceedings of the Linix Symposium. 2007.

[24] D. Price and A. Tucker. 2004. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In Proceedings of LISA '04: Eighteenth Systems Administration Conference, pages 241-254. USENIX Association.

[25] Openstack. DOI=https://www.openstack.org/

[26] Open Compute Project Reference Designs. DOI=http://www.opencompute.org/wiki/Networking/SpecsAndDesigns.

[27] Kenneth P. Birman and Thomas A. Joseph. 1987. Reliable Communication in the Presence of Failures. ACM Transactions on Computer Systems. 1987.

[28] Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks, IEEE Standard 802.1Q, 2011.

[29] Y. Luo, E. Murray, T. Ficarra. Accelerated Virtual Switching with Programmable NICs for Scalable Data Center Networking. 2010. ACM SIGCOMM 2010, pages 65-72.

[30] D. Freimuth, E. Hu, J. LaVoie, R. Mraz, E. Nahum, P. Pradhan, J. Tracey. Server Network Scalability and TCP Offload. 2005. 2005 USENIX Annual Technical Conference, pages 209-222.

[31] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Internet RFCs, ISSN 2070-1721, RFC 7348, 2014.

[32] Open vSwitch. DOI=http://openvswitch.org.

[33] vmware NSX. DOI=http://www.vmware.com/products/nsx.

[34] Open Daylight. DOI=http://www.opendaylight.org.

[35] T. Koponen, K. Amidon, et. al. Network Virtualization in Multi-tenant Datacenters. 2014. In USENIX NSDI 2014.