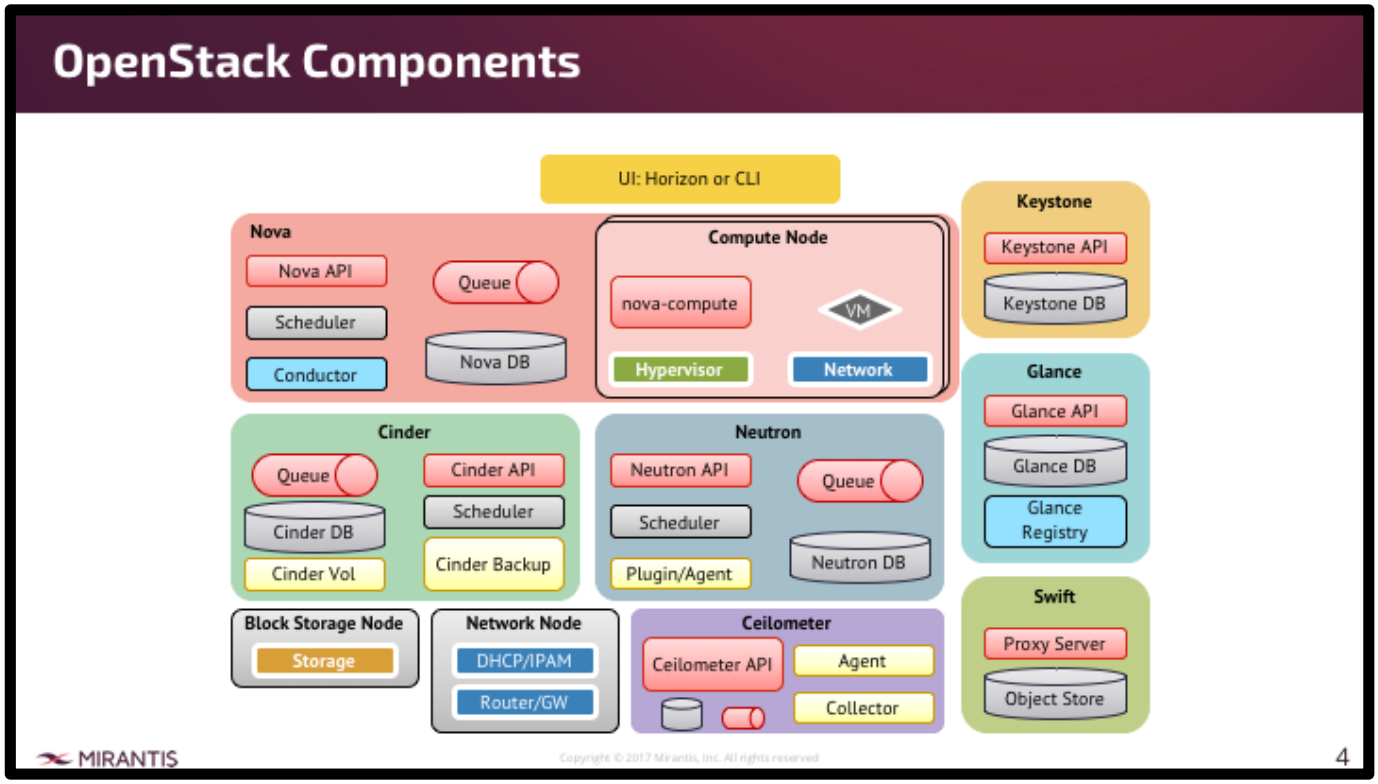

MIRANTIS
Networking Basics
An Overview
training.mirantis.com
Copyright © 2017 Mirantis, Inc. All rights reserved

Objectives

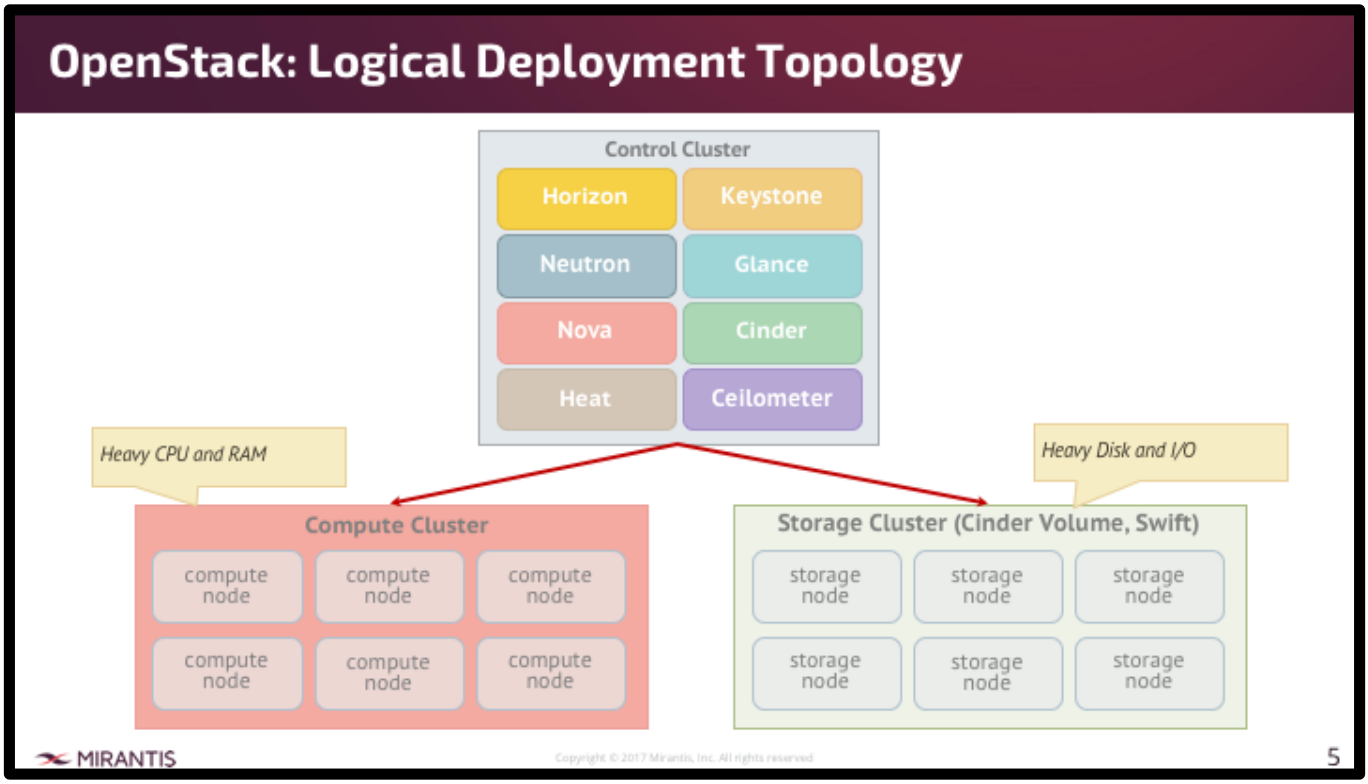
- OpenStack Networks
- Network Virtualization
  - Device Virtualization
  - Switch Virtualization
  - Overlay Networks

OpenStack Networks

Networking In The Large



1. OS is the integration of many independent “programs”.
2. Each program is comprised of several python daemons, plus a database, optional message queue.
3. The python daemons can run on the controller, compute, storage, or network nodes.



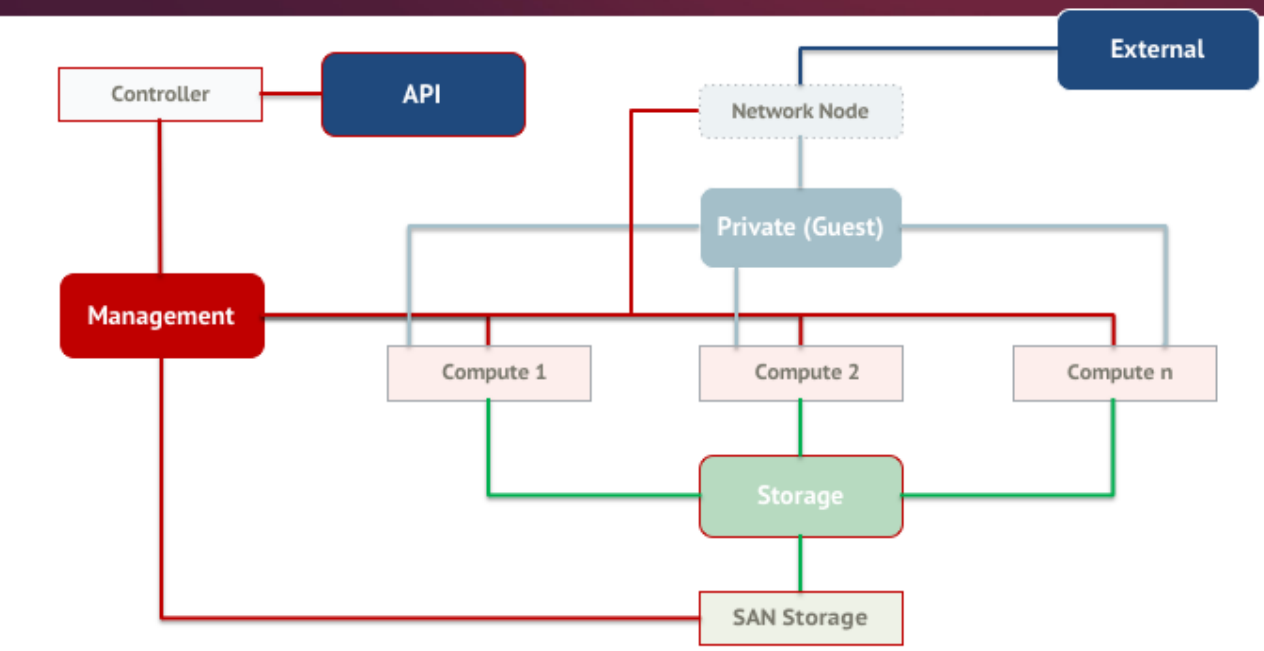
The motivation for separation is

1. The HW and resource demand is different for each service, i.e. lots of CPU and memory for compute nodes not needed on other services
2. You don't want to affect core services by move, adds, changes on compute or storage resources
3. Generally a many: one relationship between compute and storage vs. shared / platform services

- Think of it as 3 clusters within the OpenStack deployment topology
- Can be deployed on a single server or VM for lab purposes
- Production environments often segregate and customize services
- Separated for HW differences

Control Cluster is a brain

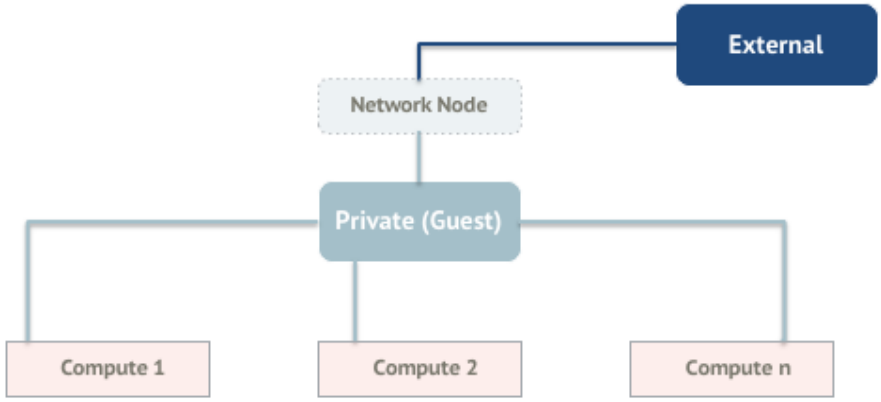
OpenStack Networks Big Picture



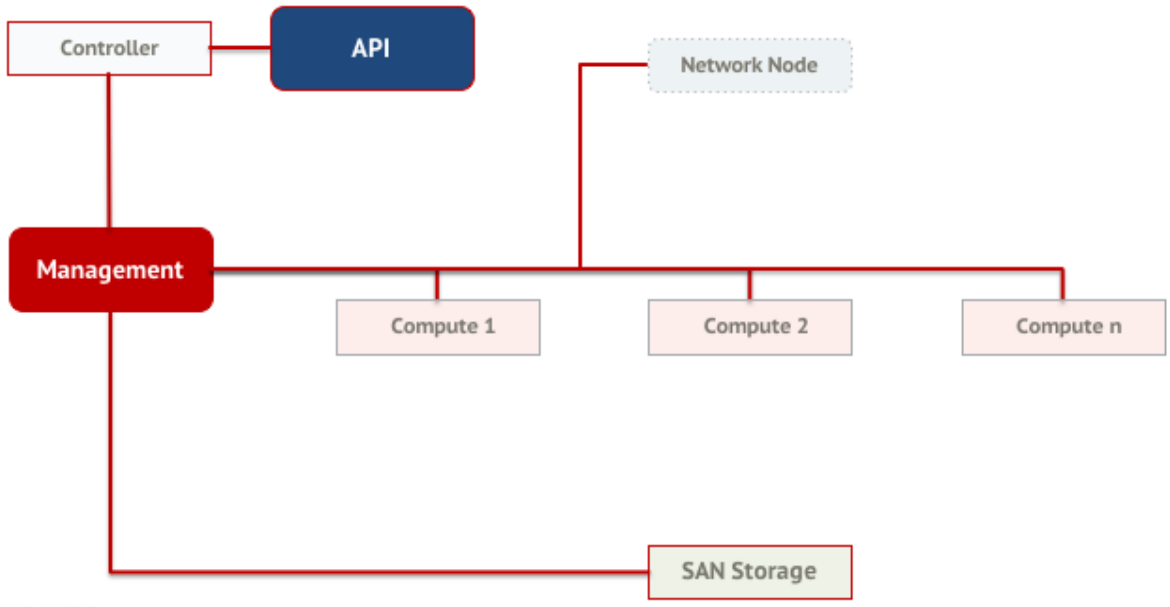
MIRANTIS

6

OpenStack Networks Managed by neutron



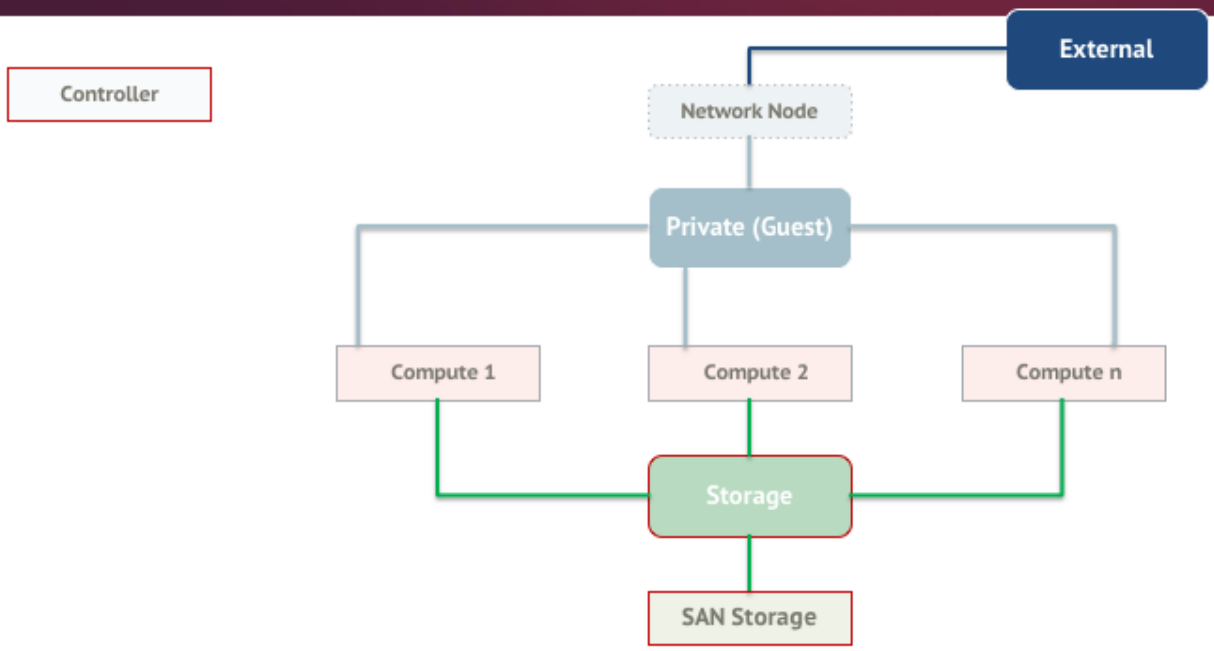
OpenStack Control Plane



MIRANTIS

8

OpenStack Data Plane



MIRANTIS

OpenStack Networking: Recap

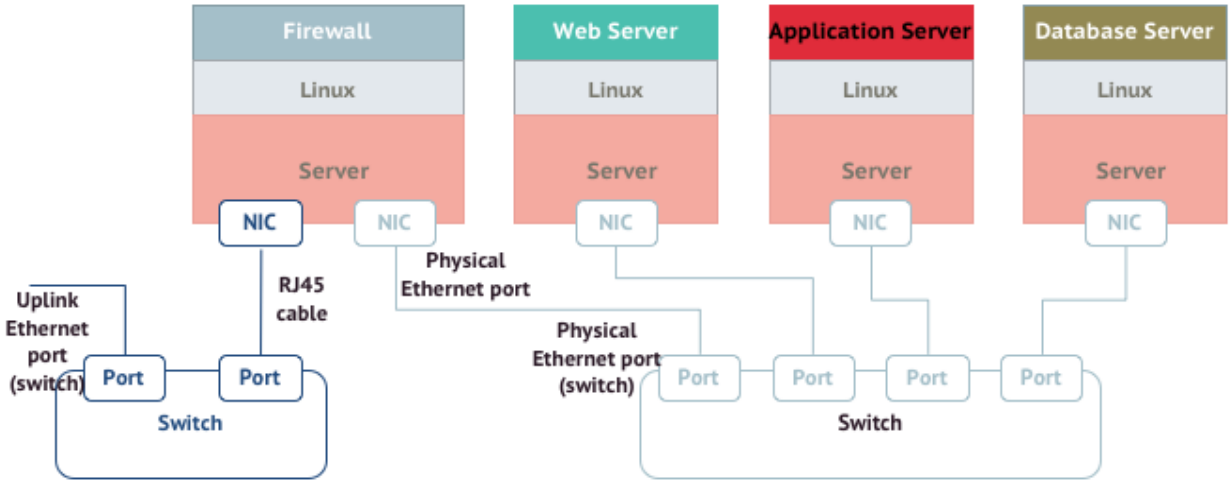
- An OpenStack installation may include the following types of networks:
  - Management network
  - Storage network
  - Private (guest) network
  - External network, API network
- Neutron only concerned with private and external networks
  - But cloud installer must consider security and bandwidth requirements for all the networks

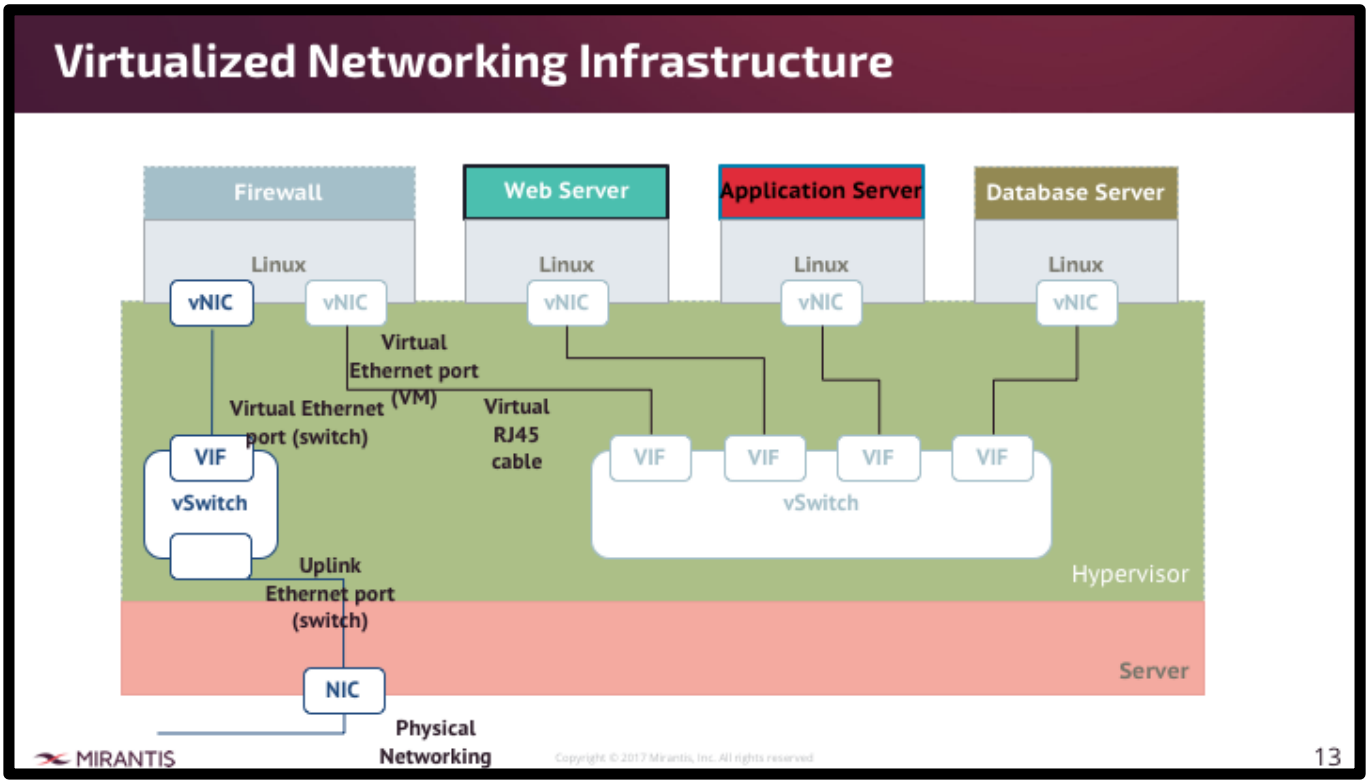
Network Virtualization

Implementation Constructs

Networking In The Small

Traditional Networking Infrastructure





Besides virtualizing a server, many network devices must be virtualized to fully map a traditional environment to an equivalent virtual environment. These devices include virtual nics, switches and ethernet cables.

Device Virtualization

vNIC: I/O SW-based sharing

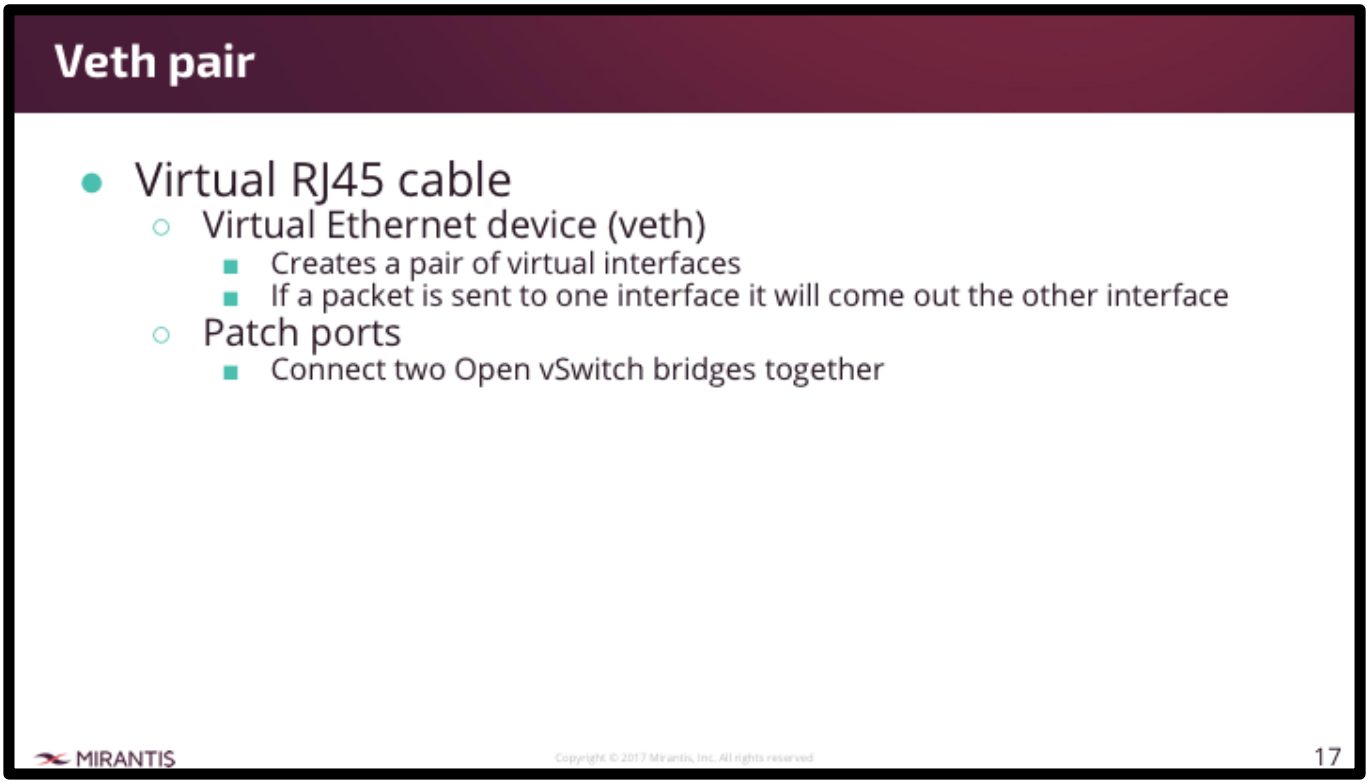
- Guest writes packets in file descriptor (FD)
- Hypervisor reads from FD and passes to host networking stack (virtual or physical)
- Device emulation – full virtualization
  - Mimics widely supported real devices (such as an Intel 1Gb NIC) and utilize existing drivers in the guest OS
  - I/O operations have to traverse two I/O stacks, one in the VM and one in the hypervisor
  - For example, QEMU
- Split-driver model – para-virtualization
  - Uses a front-end driver in the guest that works together with a back-end driver in the hypervisor
  - No need to emulate an entire device
  - For example, virtio, Xen device drivers, VMWare vmxnet

MIRANTIS 15

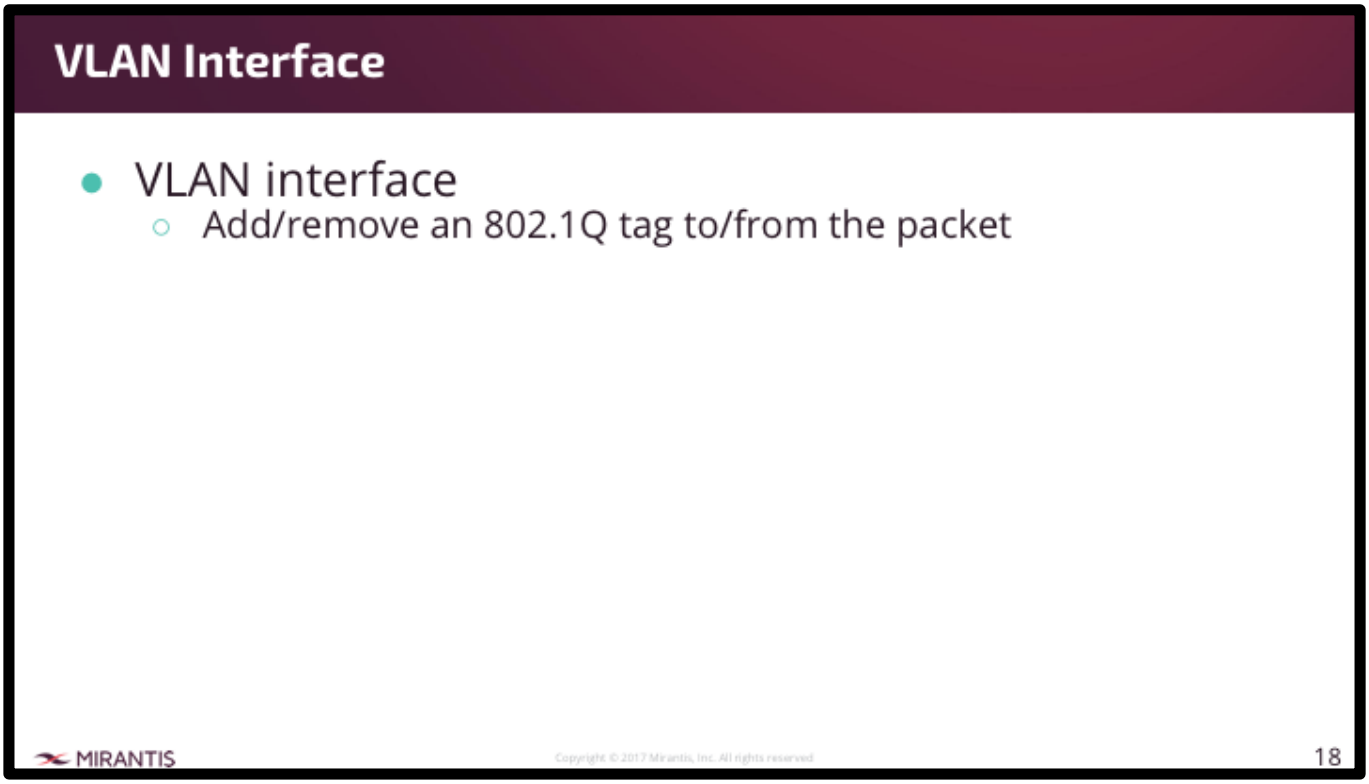
http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html

TAP

- Virtual network interface (kernel device)
  - Operates on the Ethernet frame level (L2)
  - TAP
    - Used with Linux bridges, Open vSwitch as a virtual switch port



http://stackalytics.com/report/blueprint/neutron/openvswitch-patch-port-use



http://stackalytics.com/report/blueprint/neutron/openvswitch-patch-port-use

Switch Virtualization

What is a Switch?

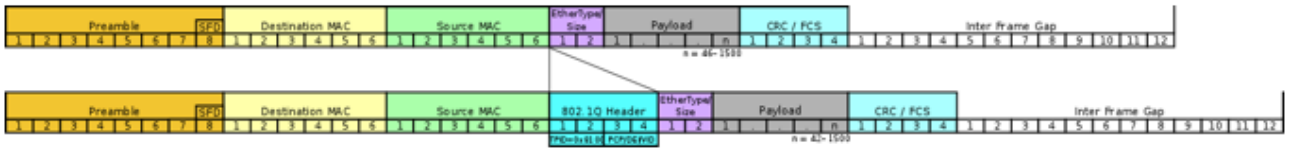
- A network switch is a device that connects devices together on a computer network, processes and forwards data at the Data link layer (L2 of OSI model)
- L2 provides a reliable link between two directly connected nodes, by detecting and possibly correcting errors that may occur in the physical layer

Traditional (ANSI/IEEE 802.1D) Switch

- Learns packet source MAC address
- Forwards the traffic to known MAC addresses
- If destination is not known, floods on all the ports except incoming – everybody gets it
- Tries to prevent packets collisions

VLAN (ANSI/IEEE 802.1Q) Switch

- Implements virtual LANs on Ethernet networks
  - Using 32-bits (4 bytes) - 12-bits of which are for Vlan ids
  - Other bits for quality-of-service prioritization (IEEE 802.1p)
- Untagged frames treated as native-vlan
  - By default vlan id 1

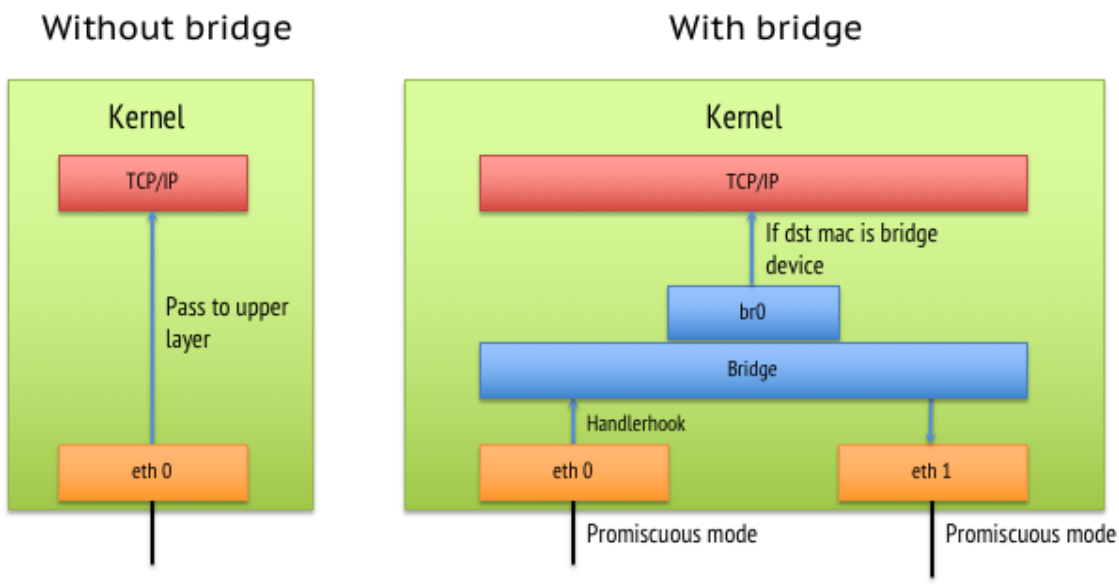


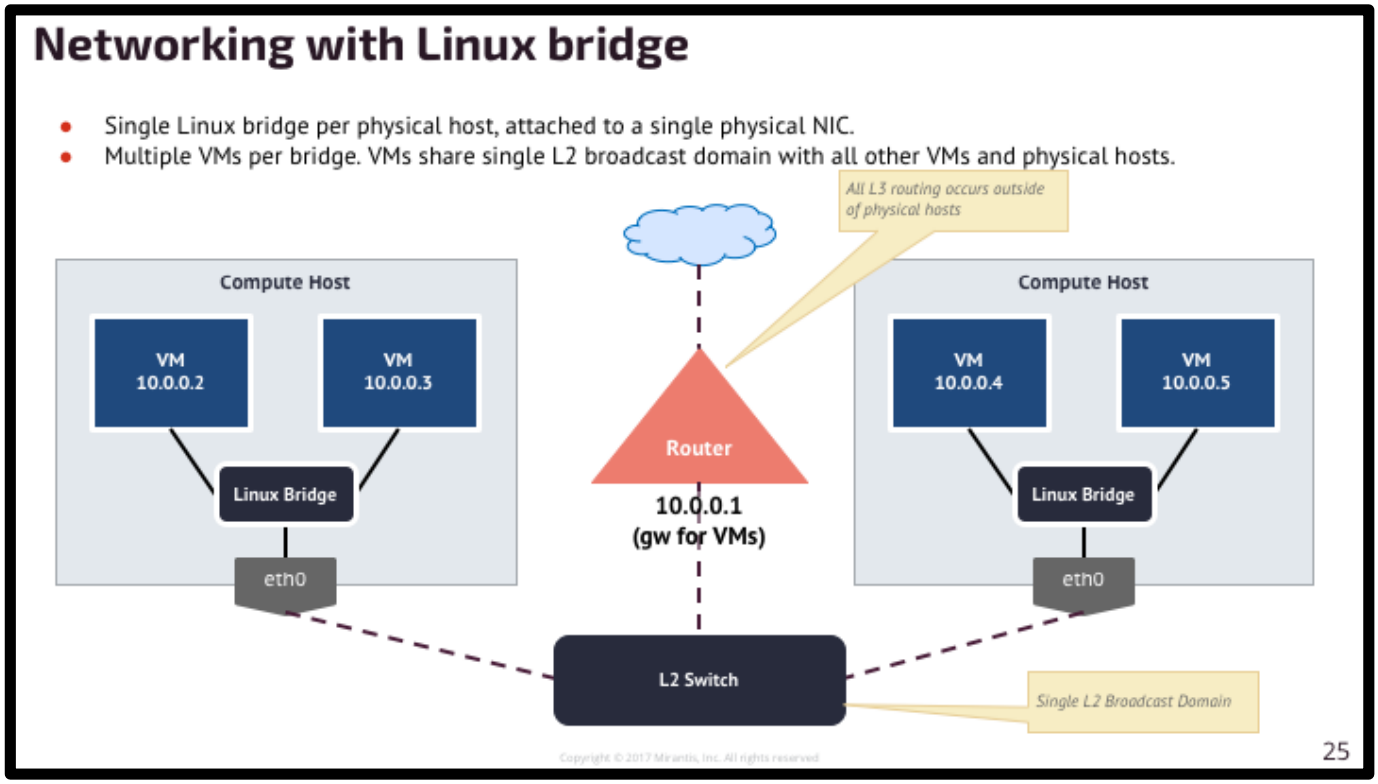
https://en.wikipedia.org/wiki/IEEE_802.1Q
QOS: https://en.wikipedia.org/wiki/IEEE_P802.1p
SPB: https://en.wikipedia.org/wiki/IEEE_802.1aq

Linux Bridge

- Open source L2 virtual switch for Linux
  - connects more than one LAN segment at Layer-2
- Implements subset of 802.1D standard
  - Supports forwarding database (FDB), spanning tree protocol (STP)
- Integrated into kernel 2.4
- Can be created:
  - In network configuration files
  - Or using “brctl” CLI:
    - brctl addbr br0
    - brctl addif br0 eth0

Linux Bridge





Before there was OpenStack, there was the Linux Bridge.
It could be used to bridge VMs together in a virtual layer-2 segment, and optionally bridge them to a physical interface, connected to a physical switch – and from there to a router.
A Linux bridge is a layer 2 device that allows multiple ethernet segments to be connected together

Open vSwitch

- Open source multi-layer (L2-L4) virtual switch
- Released in 2011, Integrated into kernel 2.6.32,
- Includes kernel and user space components
- Supports:
  - port mirroring and link aggregation
  - Per VM interface traffic policing, Fine-grained QoS control
  - IEEE 802.1D source learning (FDB)
  - IEEE 802.1Q VLAN
  - OpenFlow protocol support (extensions for virtualization)
  - Multiple overlay protocols (VxLAN, GRE, IPsec, STT, GENEVE)

Open vSwitch For Windows

- Open vSwitch 2.5
  - Uses Hyper-V datapath for kernel portion
- Hyper-V virtual switch
  - L2 switch
  - Supports IEEE 802.1Q

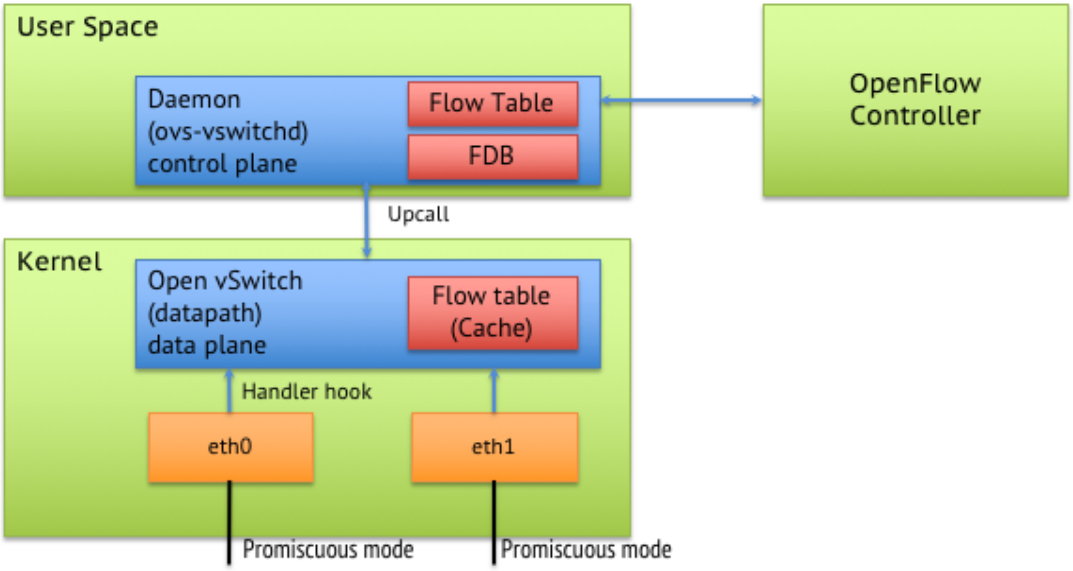
http://superuser.openstack.org/articles/tutorial-open-vswitch-hyper-v-openstack/
https://cloudbase.it/installing-openstack-nova-compute-on-hyper-v/

OpenvSwitch Components

- `ovs-vswitchd`, the daemon that implements the switch, along with a companion Linux kernel module for flow-based switching
- `ovs-vsctl`, utility to query/update `ovs-vswitchd` config
- `ovs-ofctl`, utility to query/control OpenFlow switches and controllers
- `ovs-appctl`, a utility that sends commands to running OpenvSwitch daemons
- `ovs-dpctl`, tool for configuring the switch kernel module
- `ovsdb-server`, a lightweight database server that ovs-vswitchd queries to obtain its configuration

http://git.openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=README;hb=HEAD

Open vSwitch



Overlay Networks

VLAN vs. Overlay Networks Scalability

- Segment the network using L3 packets
  - Referred to as L2 over L3

NETWORK TYPE	NUMBER OF BITS	MAX NUMBER OF TENANT NETWORKS
VLAN	12	4096
VxLAN	24	16,777,216
GRE	32	4,294,967,296
STT	64	Virtually unlimited

https://tools.ietf.org/html/draft-davie-stt-01

What is VXLAN?

- As described in RFC 7348:

  Virtual eXtensible Local Area Network (VXLAN) [...] is used to address the need for overlay networks within virtualized data centers accommodating multiple tenants. The scheme and the related protocols can be used in networks for cloud service providers and enterprise data centers.

https://tools.ietf.org/html/rfc7348

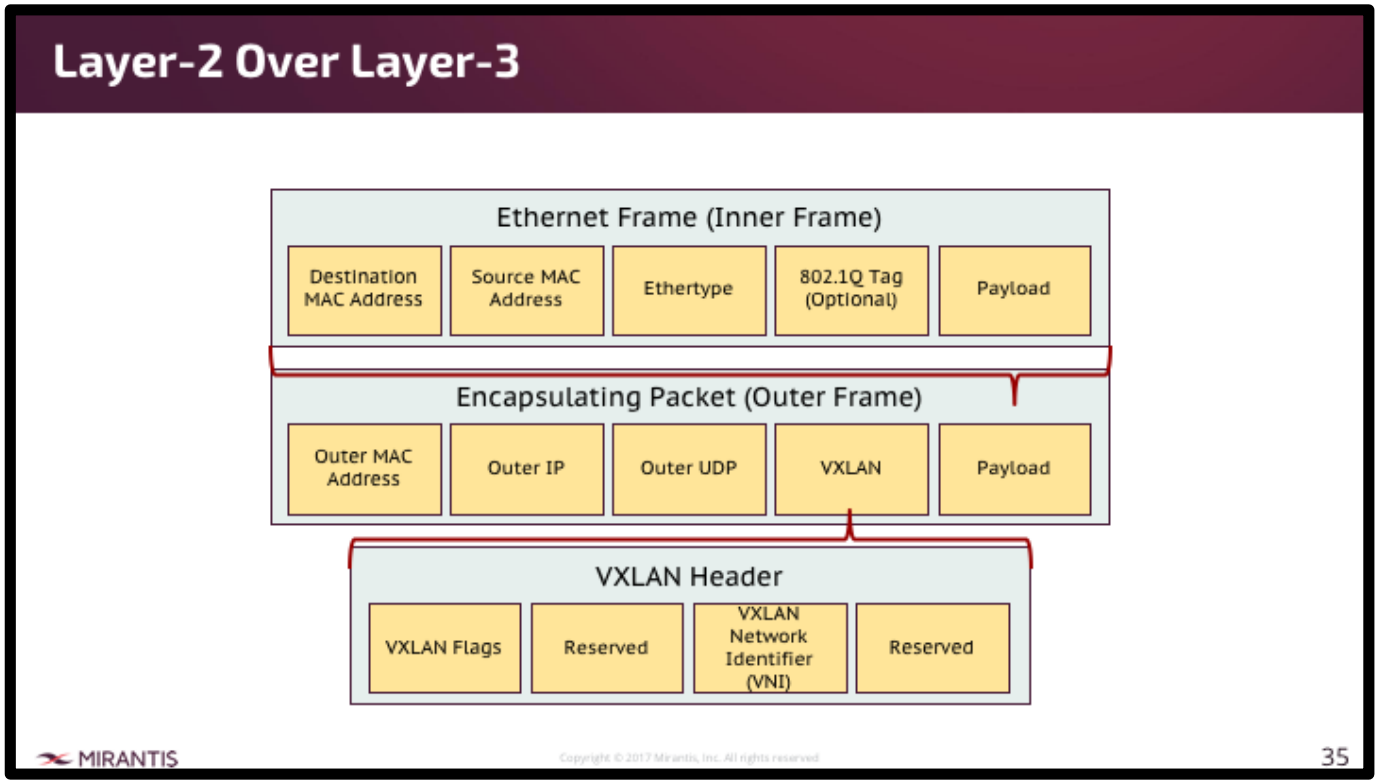
What is VXLAN?

- Tunneling protocol initially developed by Cisco, Arista, VMware, and others
- Lays on top of the physical network - also known as a *network overlay*
- Designed to use existing networking infrastructure
- Expands a single layer 2 network over layer 3
- Ideal for virtualized environments
  - Allows for dynamic changes to be performed programmatically
  - Underlying physical network need not be changed/re-configured

Why VXLAN?

- Addresses limitations of STP and VLAN ranges
  - 12-bit field for VLAN sets upper limit of tenant networks to 4094
- Designed from the ground up with multi-tenant environments in mind
- Addresses inadequate table sizes at top of the rack (ToR) switch
  - In virtualized environments, ToR switch(es) need to maintain table entries for MAC addresses for all physical and virtual servers
  - If tables overflow, switch may stop learning addresses causing significant flooding of subsequent frames

https://tools.ietf.org/html/rfc7348#section-3



Outer MAC Address: The outer destination MAC address in this frame may be the address of the target VTEP or of an intermediate Layer 3 router.

Outer IP Header: This is the outer IP header with the source IP address indicating the IP address of the VTEP over which the communicating VM (as represented by the inner source MAC address) is running. The destination IP address can be a unicast or multicast IP address (see Sections 4.1 and 4.2). When it is a unicast IP address, it represents the IP address of the VTEP connecting the communicating VM as represented by the inner destination MAC address. For multicast destination IP addresses, please refer to the scenarios detailed in Section 4.2.

Outer UDP Header: This is the outer UDP header with a source port provided by the VTEP and the destination port being a well-known UDP port.

- Destination Port: IANA has assigned the value 4789 for the VXLAN UDP port, and this value SHOULD be used by default as the destination UDP port. Some early implementations of VXLAN have used other values for the destination port. To enable interoperability with these implementations, the destination port SHOULD be configurable.
- Source Port: It is recommended that the UDP source port number be calculated using a hash of fields from the inner packet -- one example being a hash of the inner Ethernet frame's headers. This is to enable a level of entropy for the ECMP/load-balancing of the VM-to-VM traffic across the VXLAN overlay. When calculating the UDP source port number in this manner, it is

RECOMMENDED that the value be in the dynamic/private port range 49152-65535 [RFC6335].

- UDP Checksum: It SHOULD be transmitted as zero. When a packet is received with a UDP checksum of zero, it MUST be accepted for decapsulation. Optionally, if the encapsulating end point includes a non-zero UDP checksum, it MUST be correctly calculated across the entire packet including the IP header, UDP header, VXLAN header, and encapsulated MAC frame. When a decapsulating end point receives a packet with a non-zero checksum, it MAY choose to verify the checksum value. If it chooses to perform such verification, and the verification fails, the packet MUST be dropped. If the decapsulating destination chooses not to perform the verification, or performs it successfully, the packet MUST be accepted for decapsulation.

VXLAN Header: This is an 8-byte field that has:

- Flags (8 bits): where the I flag MUST be set to 1 for a valid VXLAN Network ID (VNI). The other 7 bits (designated "R") are reserved fields and MUST be set to zero on transmission and ignored on receipt.
- VXLAN Segment ID/VXLAN Network Identifier (VNI): this is a 24-bit value used to designate the individual VXLAN overlay network on which the communicating VMs are situated. VMs in different VXLAN overlay networks cannot communicate with each other.
- Reserved fields (24 bits and 8 bits): MUST be set to zero on transmission and ignored on receipt.

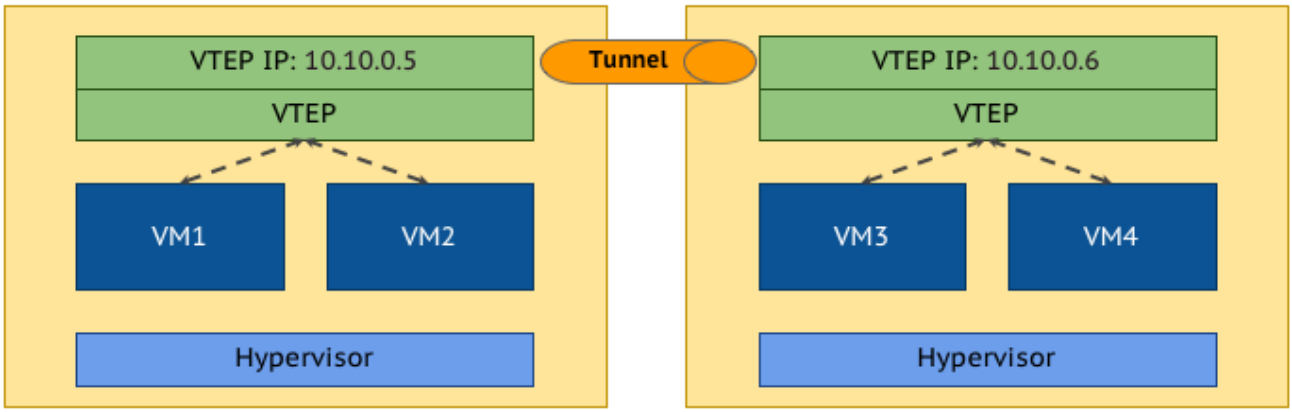
Reference

Optional Material

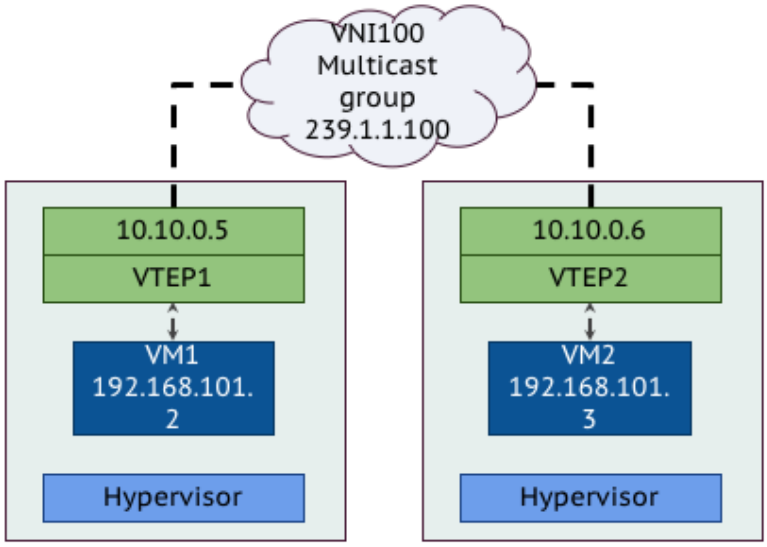
Key Concepts

- Virtual Tunnel End Point (VTEP)
  - Network device that corresponds to the beginning or end of the tunnel
  - Owns IP address that will be used in network communication
  - Can reside on a virtual switch, physical switch, firewalls, routers, etc.
  - Responsible for performing encapsulation and decapsulation packets for the different VNIs
- Virtual Network Identifier (VNI)
  - 24 bit field used to identify a particular VXLAN segment
  - Analogous to a VLAN ID in the traditional networking world

How does it work?



How does it work?



- VM1 doesn't know the MAC address for VM2 yet - sends an ARP broadcast packet
- VTEP1 receives ARP packet
- VTEP1 encapsulates packet into a multicast packet to group 239.1.1.100
- All VTEPs subscribed to the multicast group receive the packet
- Each VTEP decapsulates the packet to look at VNI the packet was placed on (VNI100)
- If that VNI is in use on that system, the VTEP forwards the packet to VMs on that VNI
- VTEP2 creates a record in local VXLAN table mapping VM1's MAC to VTEP1 (10.10.0.5)
- VM2 receives ARP packet and sends back a response to VM1
- VTEP2 encapsulates response and sends back to VTEP1 (10.10.0.5)
- VTEP1 receives response, decapsulates the packet and forwards to VM1
- VTEP1 creates a record in local VXLAN table mapping VM2's MAC to VTEP2 (10.10.0.6)

MAC Learning Behavior

- When VMs on different nodes look to communicate, it is up to the VTEPs to handle the communication:
  - If the VTEP has an entry for the destination MAC in its table, then it will pass the packet to the VTEP associated with that MAC
  - If the VTEP does not have an entry for the destination MAC in its table, then it will flood the packet to all VTEPs in the multicast group
- Flooding is expensive and does not scale well