# Linux Bridge, l2-overlays, E-VPN!

Roopa Prabhu
Cumulus Networks

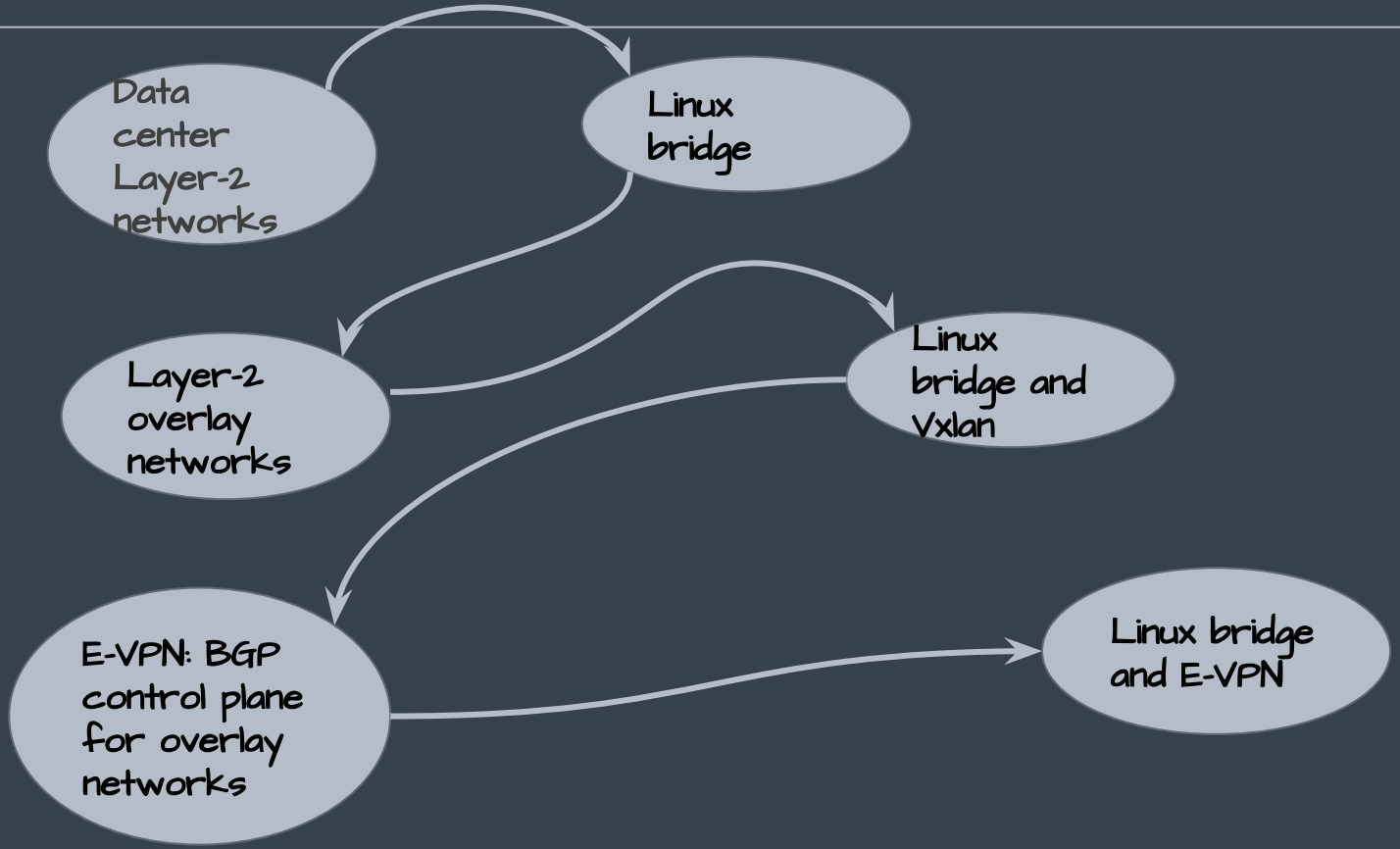
# This tutorial is about ...

- Linux bridge at the center of data center Layer-2 deployments
- Deploying Layer-2 network virtualization overlays with Linux
- Linux hardware vxlan tunnel end points
- Ethernet VPN's: BGP as a control plane for Network virtualization overlays

# Tutorial Focus/Goals ..

- Outline and document Layer-2 deployment models with Linux bridge
- Focus is on Data center deployments
  - All Examples are from a TOR (Top-of-the-rack) switch running Linux Bridge

# Tutorial flow ...



# Data Center Network Basics

- Racks of servers grouped into PODs
- Vlans, Subnets stretched across Racks or POD's
- Overview of data center network designs [1]
  - Layer 2
  - Hybrid layer 2-3
  - Layer 3
- Modern Data center networks:
  - Clos Topology [2]
  - Layer 3 or Hybrid Layer 2-3

# Modern Data center network

# Hybrid layer-2 - layer-3 data center network

# Layer-3 only data center network

# Layer-2 Gateways with Linux Bridge

# Layer-2 Gateways with Linux Bridge

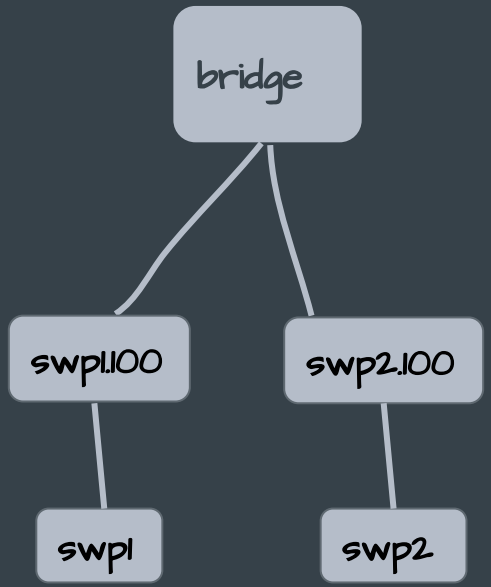
- Connect layer-2 segments with bridge
- Bridge within same vlans
- TOR switch can be your L2 gateway bridging between vlans on the servers in the same rack

# What do you need ?

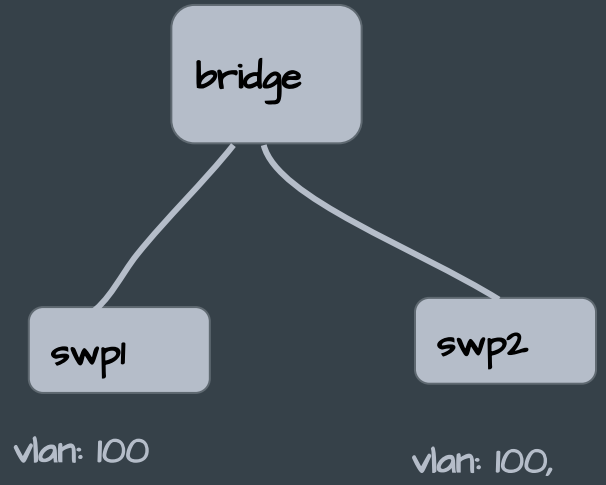
- TOR switches running Linux Bridge
- Switch ports are bridge ports
- Bridge vlan filtering or non-vlan filtering mode:
    - Linux bridge supports two modes:
        - A more modern scalable vlan filtering mode
        - Or old traditional non-vlan filtering mode

# Layer-2 switching within a vlan

## Non-vlan filtering bridge

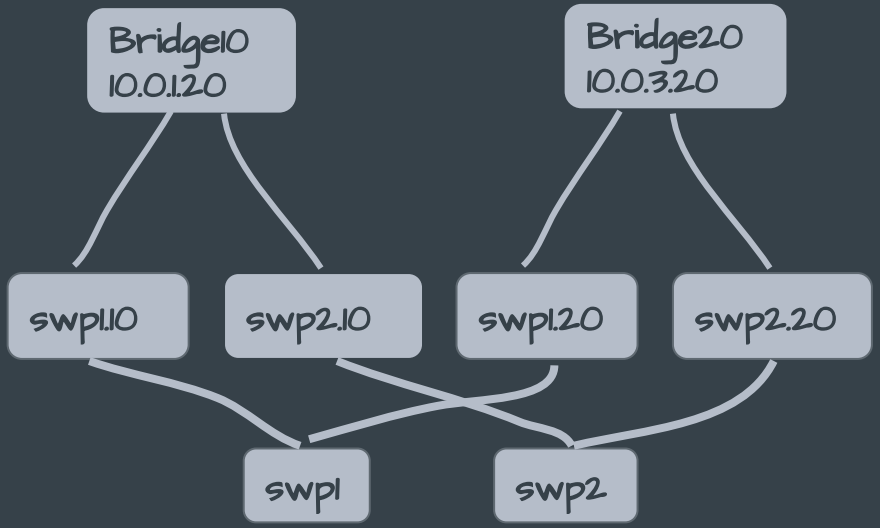



## vlan filtering bridge

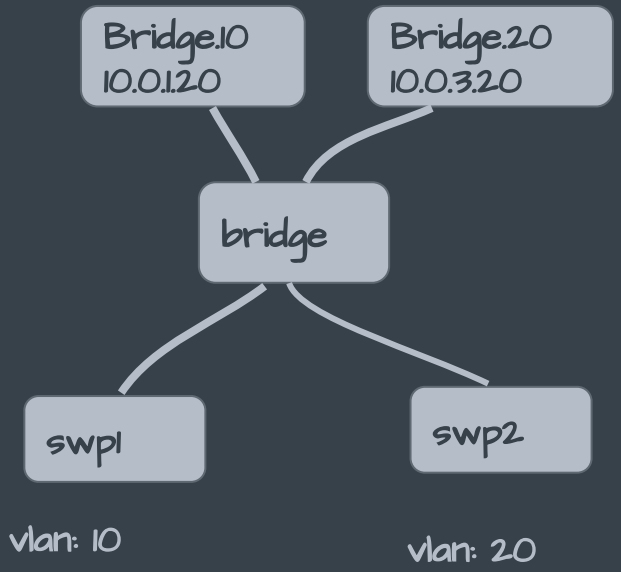


# Routing between vlans

## Non-vlan filtering bridge




## vlan filtering bridge



# Scaling with Linux bridge

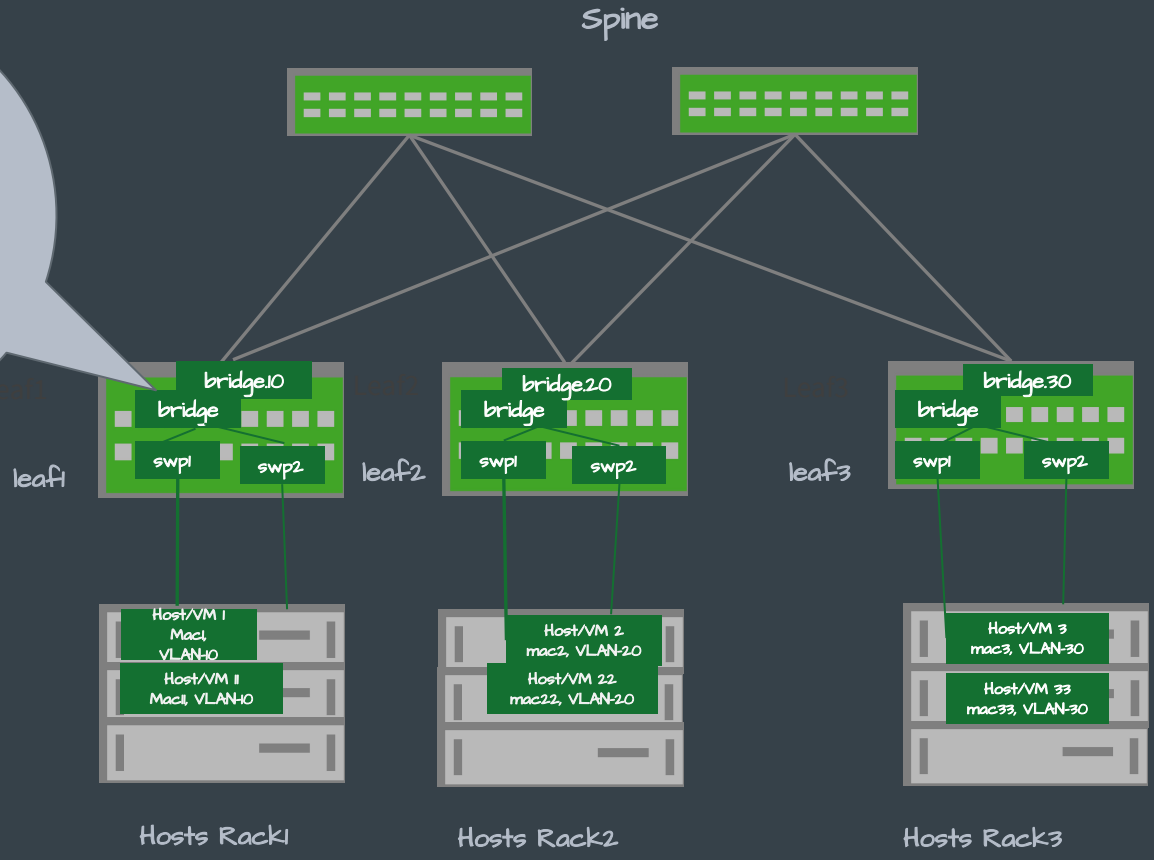
A vlan filtering bridge results in less number of overall net-devices

Example: Deploying 2000 vlans 1-2000 on 32 ports:

- non-Vlan filtering bridge:
  - Ports + 2000 vlan devices per port + 2000 bridge devices
  - 32 + 2000 * 32 + 2000 = 66032
- Vlan filtering bridge:
  - 32 ports + 1 bridge device + 2000 vlan devices on bridge for routing
  - 32 + 1 + 2000 = 2033 netdevices

# L2 gateway on the TOR with Linux bridge

- leaf* are l2 gateways
- Bridge within the same vlan and rack and route between vlans
- bridge.* vlan interfaces are used for routing



# Bridge features and flags

- Learning
- Igmp Snooping
- Selective control of broadcast, multicast and unknown unicast traffic
- Arp and ND proxying
- STP

Note: for the rest of this tutorial we will only use the vlan filtering bridge for simplicity.

# Layer-2 - Overlay Networks

# Overlay networks basics

- Overlay networks are an approach for providing network virtualization services to a set of Tenant Systems (TSs)
- Overlay networks achieve network virtualization by overlaying layer 2 networks over physical layer 3 networks

# Network Virtualization End-points

- Network virtualization endpoints (NVE) provide a logical interconnect between Tenant Systems that belong to a specific Virtual network (VN)

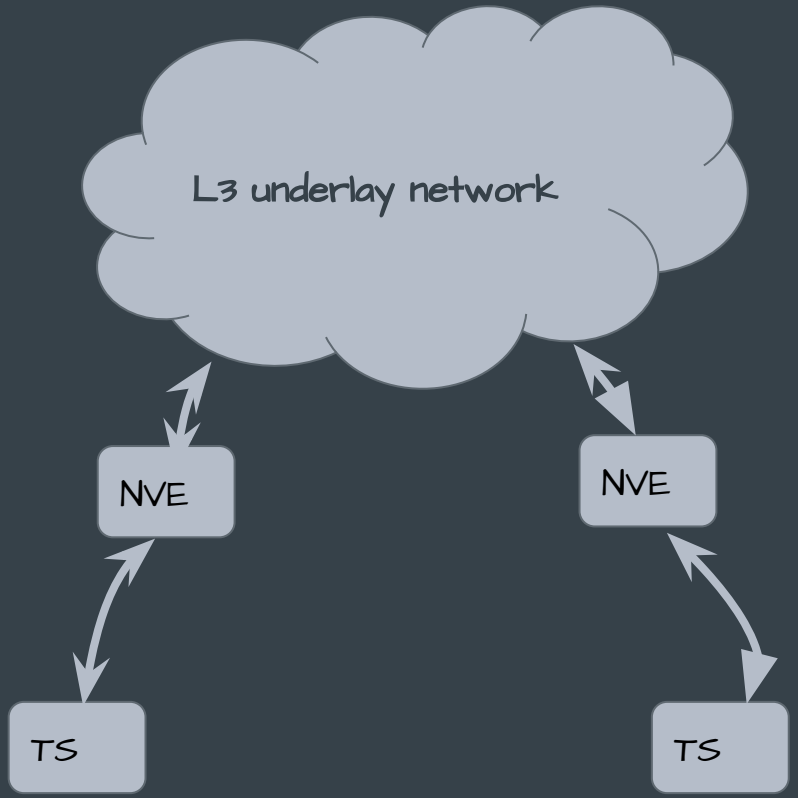
- NVE implements the overlay protocol (eg: vxlan)

# NVE Types

- Layer-2 NVE
  - Tenant Systems appear to be interconnected by a LAN environment over an L3 underlay

- Layer-3 NVE
  - An L3 NVE provides virtualized IP forwarding service, similar to IP VPN

# Overlay network



# Why Overlay networks ?

- Isolation between tenant systems
- Stretch layer-2 networks across racks, POD's, inter or intra data centers
  - Layer-2 networks are stretched
    - To allow VM's talking over same broadcast domain to continue after VM mobility without changing network configuration
    - In many cases this is also needed due to Software licensing tied to mac-addresses

# Why Overlay networks ? (Continued)

- Leverage benefits of L3 networks while maintaining L2 reachability
- Cloud computing demands:
  - Multi tenancy
  - Abstract physical resources to enable sharing

# NVE deployment options

Overlay network end-points (NVE) can be deployed on

- The host or hypervisor or container OS (System where Tenant systems are located)

  OR

- On the Top-of-the-rack (TOR) switch

# VTEP on the servers or the TOR ?

Vxlan tunnel endpoint on the servers:

- Hypervisor or container orchestration systems can directly map tenants to VNI
- Works very well in a pure layer-3 datacenter: terminate VNI on the servers

Vxlan tunnel endpoint on the TOR:

- A TOR can act as a l2 overlay gateway mapping tenants to VNI
- Vxlan encap and decap at line rate in hardware
- Tenants are mapped to vlans. Vlans are mapped to VNI at TOR

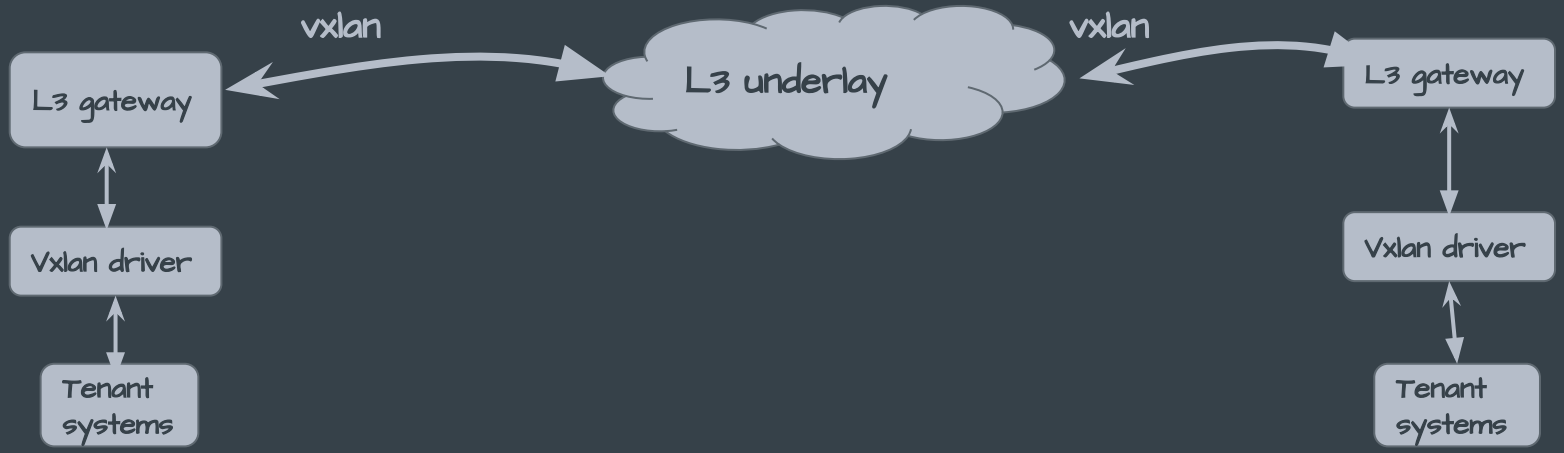
# Layer-2 Overlay network dataplane: vxlan

- VNI - virtual network identifier (24 bit)
- Vxlan tunnel endpoints (VTEPS) encap and decap vxlan packets
- VTEP has a routable ip address
- Linux vxlan driver
- Tenant to vni mapping

# Vxlan tunnel end-point on the hypervisor

# Vxlan tunnel end-point on the TOR switch

# Linux vxlan tunnel end point (layer-3)




- Tenant systems directly mapped to VNI

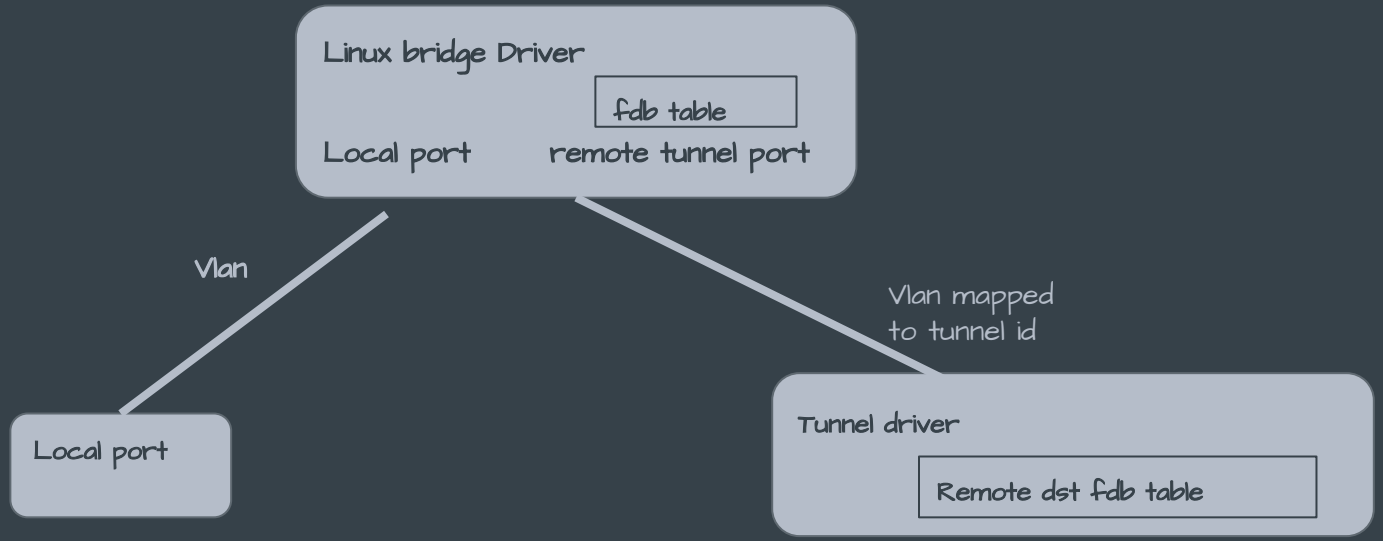
# Linux Layer-2 overlay gateway: vxlan


- Tenant systems mapped to vlans
- Linux bridge on the TOR maps vlans to vni

# FDB Learning options

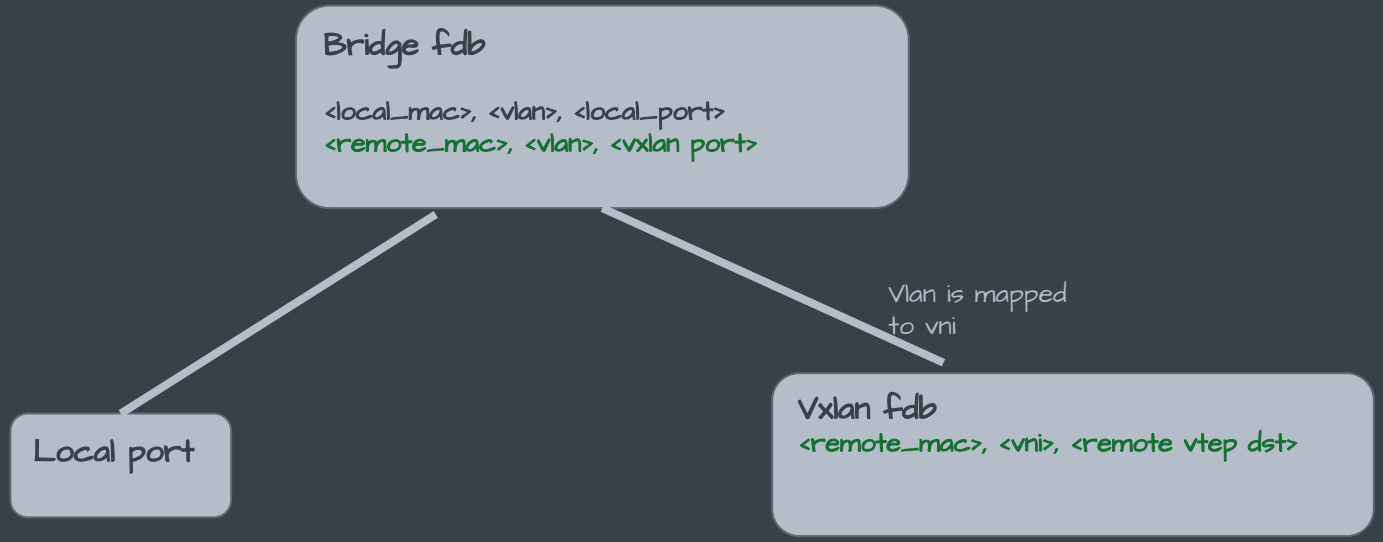
- Flood and learn (default)
- Control plane learning
  - Control plane protocols disseminate end point address mappings to vteps
  - Typically done via a controller
- Static mac install via orchestration tools

# Layer-2 overlay gateway tunnel fdb tables




- Linux bridge and tunnel endpoints maintain separate fdb tables
- Linux bridge fdb table contains all macs in the stretched L2 segment
- Tunnel end point fdb table contains remote dst reachability information

# Bridge and vxlan driver fdb




- Vxlan fdb is an extension of bridge fdb table with additional remote dst entry info per fdb entry
- Vlan entry in bridge fdb entry maps to vni in vxlan fdb

# Broadcast, unknown unicast and multicast traffic (BUM)



- An l2 network by default floods unknown traffic
- Unnecessary traffic leads to wasted bw and cpu cycles
- This is aggravated when l2 networks are stretched to larger areas: across racks, POD's or data centers
- Various optimizations can be considered in such l2 stretched overlay networks

# Bridge driver handling of BUM traffic

Bridge driver has separate controls

- To drop broadcast, unknown unicast and multicast traffic

# Vxlan driver handling of BUM traffic

- Multicast:
  - Use a multicast group to forward BUM traffic to registered vteps
    - Multicast group can be specified during creation of the vxlan device
- Head end replication:
    - Default remote vtep list to replicate a BUM traffic to
    - Can be specified by a vxlan all-zero fdb entry pointing to a remote vtep list
- Flood: simply flood to all remote ports
  - Control plane can minimize flood by making sure every vtep knows remote end-points it cares about
-

# Vxlan netdev types

- A traditional vxlan netdev
  - Deployed with one netdev per vni
  - Each vxlan netdev maintains forwarding database (fdb) for its vni
    - Fdb entries hashed by mac
- Recent kernels support ability to deploy a single vxlan netdev for all VNI's
  - Such a mode is called collect_metadata or LWT mode
  - A single forwarding database (fdb) for all VNI's
  - Fdb entries are hashed by <mac, VNI>

# Linux L2 vxlan overlay gateway example

# Building an Linux l2 overlay gateway

- Vxlan tunnel netdevice(s) for encap and decap
- Linux bridge device with local and vxlan ports
- Bridge maps Vlan to vni
- Bridge switches
  - Vlan traffic from local to remote vxlan ports
  - Remote traffic from vxlan ports to local vlan ports

# Recipe-1

[one vxlan netdev per vni

Example shows leaf1 config]

# Recipe 1: create all your netdevs

```
$ # create bridge device:
$ ip link add type bridge dev bridge
$ # create vxlan netdev:
$ ip link add type vxlan dev vxlan-10 vni 10 local 10.1.1.1
$ # enslave local and remote ports
$ ip link set dev vxlan-10 master bridge
$ ip link set dev swp1 master bridge
```

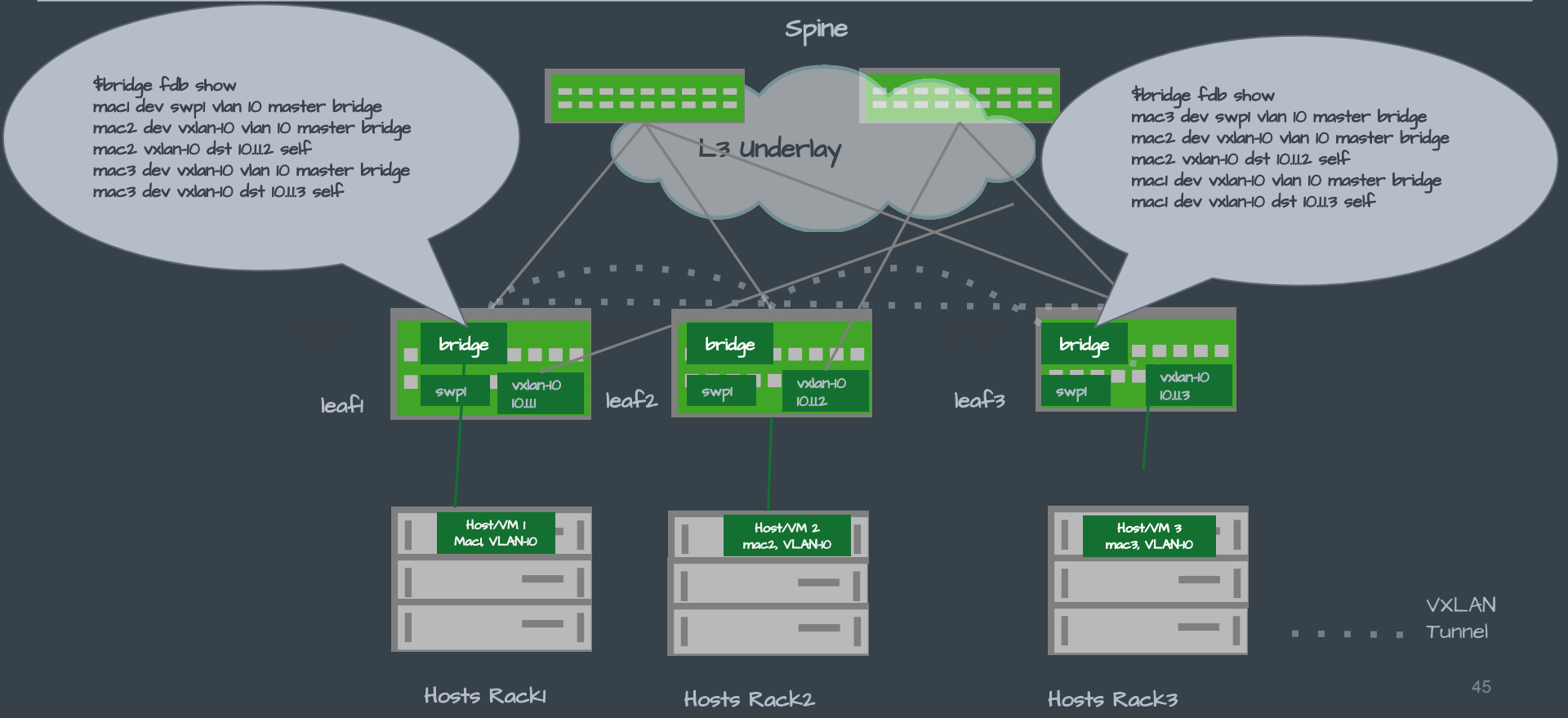
# Recipe 1: Configure vlan filtering and vlans

```
$ #configure vlan filtering on bridge
$ ip link set dev bridge type bridge vlan_filtering 1
$ #configure vlans
$ bridge vlan add vid 10 dev vxlan-10
$ bridge vlan add vid 10 untagged pvid dev vxlan-10
$ bridge vlan add vid 10 dev swp1
```

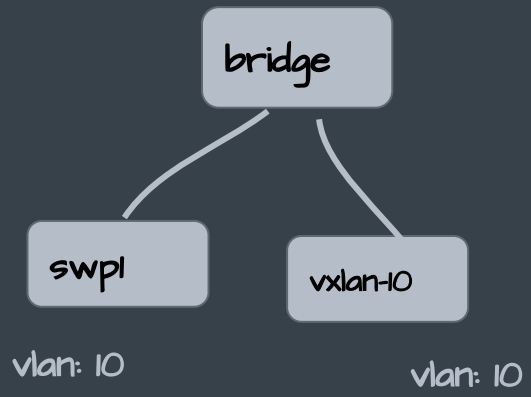
# Recipe 1: Add default fdb entries

```
$ # add your default remote dst forwarding entry
$ bridge fdb add 00:00:00:00:00:00 dev vxlan-10 dst 10.1.1.2
  self permanent
$ bridge fdb add 00:00:00:00:00:00 dev vx-10 dst 10.1.1.3
  self permanent
```

# Recipe 1: Here's how it all looks



# Zoom into the bridge config on the TOR switches

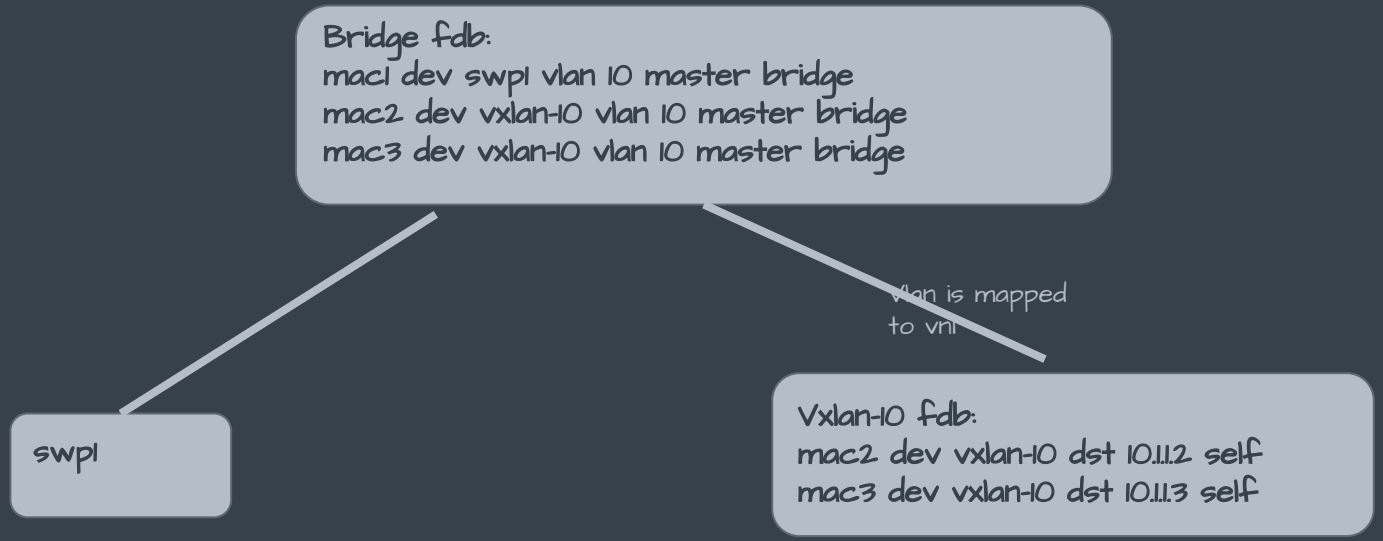



```
$bridge vlan show
port     vlan ids
swp1     1 PVID Egress Untagged
         10

vxlan-10   10 PVID Egress Untagged
           10
```

- Vlan 10 is mapped to vxlan vni 10

# Zoom into bridge and vxlan driver fdb tables




- Vxlan fdb is an extension of bridge fdb table with additional remote dst info per fdb entry
- Vlan entry in bridge fdb entry maps to vni in vxlan fdb

# Recipe 1: check your running kernel state

```
$ ip link show master bridge
$ bridge vlan show
port     vlan ids
vxlan-10  10 PVID Egress Untagged
swp1     1  PVID Egress Untagged
         10
bridge None
```

```
$ bridge fdb show
mac1 dev swp1 vlan 10 master bridge
mac2 dev vxlan-10 vlan 10 master bridge
mac2 vxlan-10 dst 10.1.1.2 self
mac3 dev vxlan-10 vlan 10 master bridge
mac3 dev vxlan-10 dst 10.1.1.3 self

$ # check bridge flags
$ ip -d link show dev bridge
```

# Recipe-2
# (With single vxlan netdev
# Example shows leaf1 config)

# Recipe 2: create all your netdevs

```
$ # create bridge device:
$ ip link add type bridge dev bridge
$ # create vxlan netdev:
$ ip link add type vxlan dev vxlan0 external local 10.0.1.1
$ # enslave local and remote ports
$ ip link set dev vxlan0 master bridge
$ ip link set dev swp1 master bridge
```

# Recipe 2: Enable vlan filtering and vlan_tunnel mode

```
$ #configure vlan filtering on bridge
$ ip link set dev bridge type bridge vlan_filtering 1


$ # enable tunnel mode on the vxlan tunnel bridge ports
$ bridge link set dev vxlan0 vlan_tunnel on
```

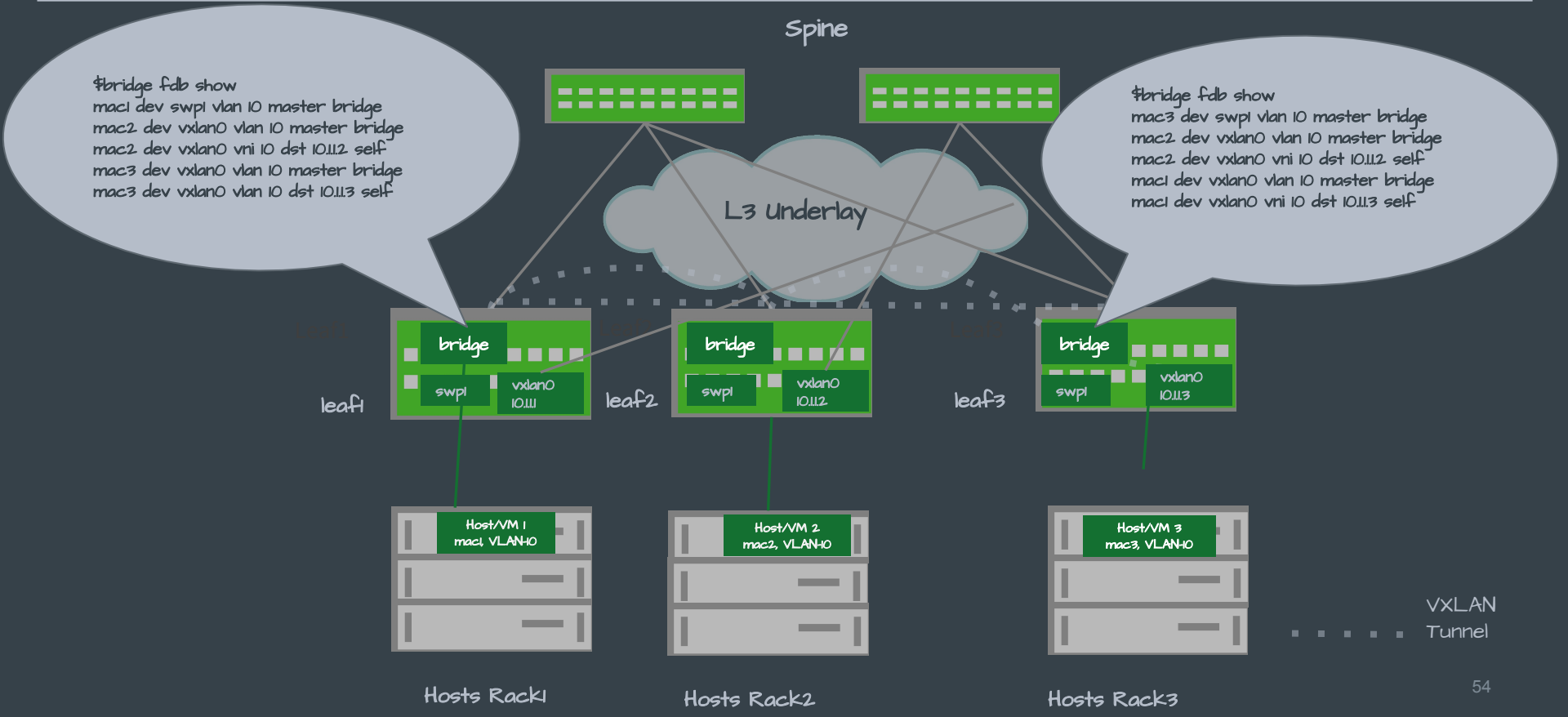
# Recipe 2: configure vlans

```
$ #configure vlans
$ bridge vlan add vid 10 dev vxlan0
$ bridge vlan add vid 10 dev swp1
$ # set tunnel mappings on the ports per vlan
$ # map vlan 10 to tunnel id 10 (in this case vni 10)
$ bridge vlan add dev vxlan0 vid 10 tunnel_info id 10
```

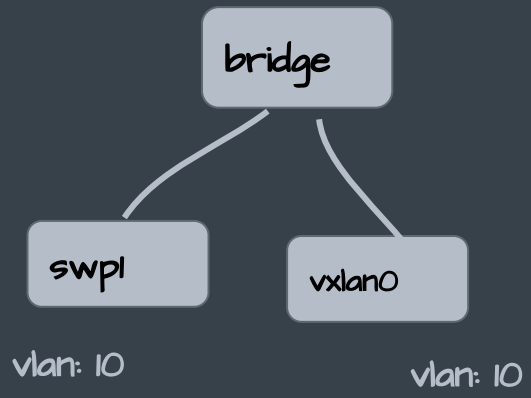
# Recipe 2: configure default fdb entries

```
$ # add your default remote dst forwarding entry
$ bridge fdb add 00:00:00:00:00:00 dev vxlan0 vni 10 dst
  10.1.1.2 self permanent
$ bridge fdb add 00:00:00:00:00:00 dev vxlan0 vni 10 dst
  10.1.1.3 self permanent
```

# Recipe 2: Here's how it all looks



# Zoom into the bridge config on the leaf switches




```
$bridge vlan show
port     vlan ids
swp1     1 PVID Egress Untagged
         10

vxlan0   1 PVID Egress Untagged
          10
```

```
$bridge vlan tunnelshow
port     vlan id   tunnel id
Vxlan0   10        10
```

- Vlan 10 is mapped to vxlan vni 10

# Recipe 2: check your running kernel state

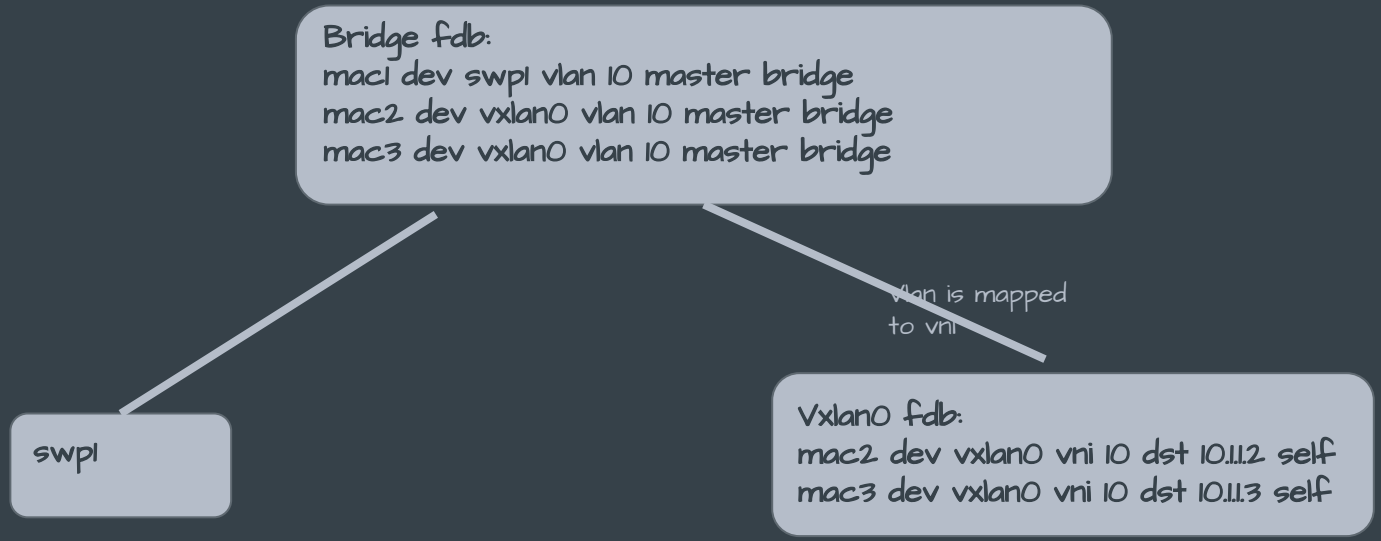
```
$ bridge vlan show
port     vlan ids
vxlan0   1 PVID Egress Untagged
                 10
swp1     1  PVID Egress Untagged
         10
bridge None
```

```
$ bridge vlan tunnelshow
port     vlan id    tunnel id
Vxlan0   10           10

$ bridge fdb show
mac1 dev swp1 vlan 10 master bridge
mac2 dev vxlan0 vlan 10 master bridge
mac2 vxlan0 dst 10.1.1.2 self
mac3 dev vxlan0 vlan 10 master bridge
mac3 dev vxlan0 dst 10.1.1.3 self

$ ip -d link show dev bridge
```

# Zoom into bridge and vxlan driver fdb tables




- Vxlan fdb is an extension of bridge fdb table with additional remote dst info per fdb entry
- Vlan entry in bridge fdb entry maps to vni in vxlan fdb

# Other Network Virtualization Technologies



- Other overlay data planes:
  - Geneve, NVGRE, STT
- ILA - Identifier Locator Addressing
  - Wise Tom Herbert says 'Move to Ipv6 and use ILA for native network virtualization' :)

# Summary overlays:

- Flood and learn by default
- Controllers can be used to disseminate MAC addresses to avoid flooding
- Distributed controllers win over Centralized controllers
- Many controller solutions available: some proprietary
- Need for an Open Standards based controller: Lets dive into the next section which does just that

# Ethernet VPNs (E-VPNs)

# What are E-VPNs ?

- Ethernet VPN i.e. another form of Layer-2 VPN
  - L2-VPN's are virtual private networks carrying layer-2 traffic
  - Different from VPLS [5, 6]
  - Used to separate tenants at Layer-2
- Original EVPN RFC 7432: [7]
  - BGP MPLS-based Ethernet VPN
  - Requirements defined in RFC 7209 [8]

# Why E-VPN ?

- Overcome limitations of prior L2-VPN technologies like VPLS
- Support for multihoming and redundancy
- Control plane learning: No flooding
- Supports multiple data plane encapsulations
- Various optimizations
  - Multicast optimization
  - ARP-ND broadcast handling

# Evpn use-cases

- Initially introduced to support l2 vpn provider services to customers
- Multi-tenant hosting
- Stretch L2 across POD's in the data center
- Data center interconnect (DCI) technology
  - Stretch l2 across data centers

In this tutorial we look at BGP based E-VPN as a distributed controller for layer-2 network virtualization

E-VPN is adopted in the data center with "vxlan" overlay. This tutorial will focus on BGP-Vxlan based E-vpn.

# New RFC's to adopt E-VPN in the data center

- A network virtualization overlay solution using E-VPN [3]
- BGP based control plane for Vxlan

# Border Gateway Protocol (BGP)

- Routing protocol of the internet
- A typical BGP implementation [10] on Linux installs routes kernel FIB to install routes
- With E-VPN, we are telling BGP to also look at layer-2 forwarding entries in the kernel and distribute to peers

# BGP E-VPN

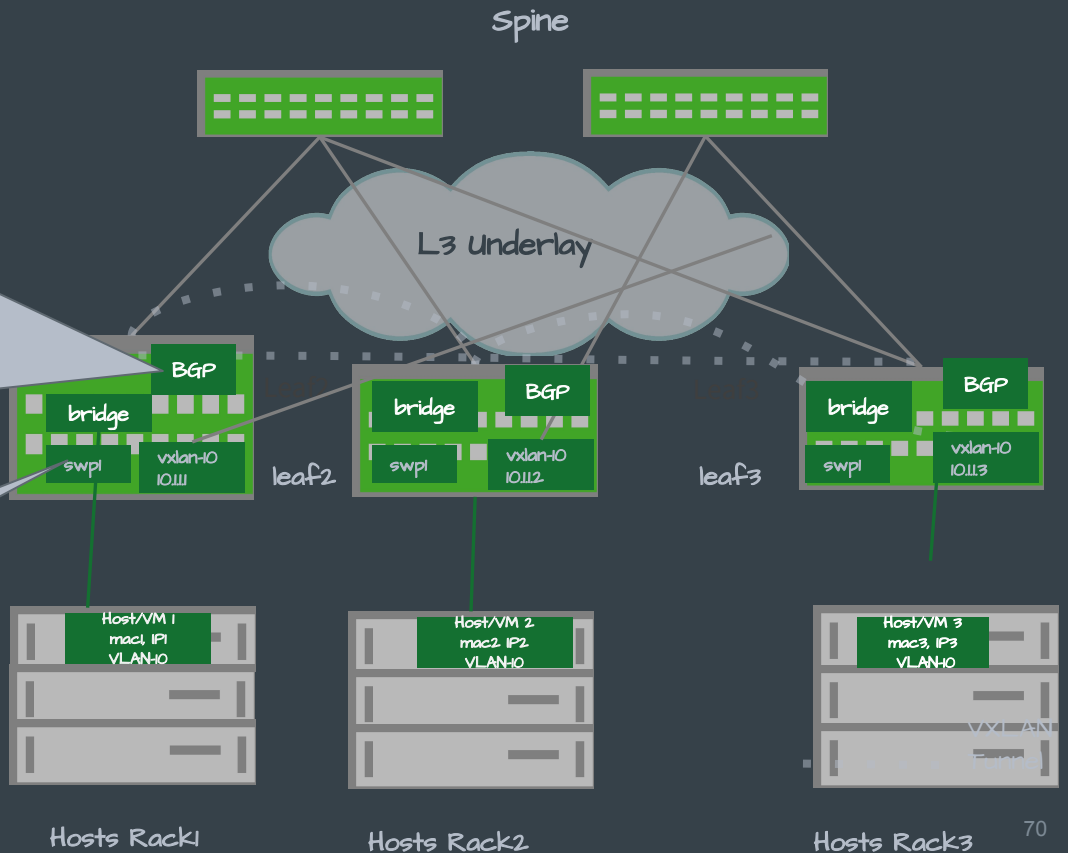
- BGP runs on each Vtep
- Peers with BGp on other Vteps
- Exchanges local Mac and Mac/IP routes with peers
- Exchanges VNI's each VTEP is interested in
- Tracks mac address moves for faster convergence
- Type of Information exchanged is tagged by 'Route types'
  - MAC or MAC-IP routes are Type 2 routes
  - BUM replication list exchanged via Type 3 routes

In this tutorial we will only focus on E-VPN on the data center TOR switches running Linux.

# E-VPN flow (distribute macs)

(a) BGP discovers local vlan-vni mapping via netlink
(b) BGP reads local bridge <mac, vlan> entries and distributes them to bgp E-vpn peers
(c) BGP learns remote <mac, vni> entries from E-VPN peers and installs them in the kernel bridge fdb table
(d) Kernel bridge fdb table has all local and remote mac's for forwarding

(a) Bridge learns local <mac, vlan> in its fdb



# Arp And ND suppression

- ARP and ND traffic is by default flooded to all nodes in the broadcast domain
- "Arp and ND suppression" is an E-VPN function
  - To reduce Arp and ND flooded traffic in such large broadcast domains
  - ARP broadcast traffic problems in large data center are described here [4]
- BGP E-VPN control plane knows remote IP-MAC's
  - These remote MAC-IP's can be used to proxy local ARP-ND requests

# Linux Bridge Arp And ND suppression for E-VPN

- BGP exchanges local MAC-IP's with E-VPN peers as Type 2 MAC-IP routes
- BGP installs remote MAC-IP's from E-VPN peers in the kernel neigh table
- Linux bridge driver uses remote MAC-IP's (neigh entries) installed by E-VPN to proxy requests for MAC-IP from local end hosts
- For a MAC-IP entry not present in the neigh table,
  - bridge driver floods such requests to all ports in that vlan/vni

# E-VPN flow: arp nd suppression (distribute mac + ip)

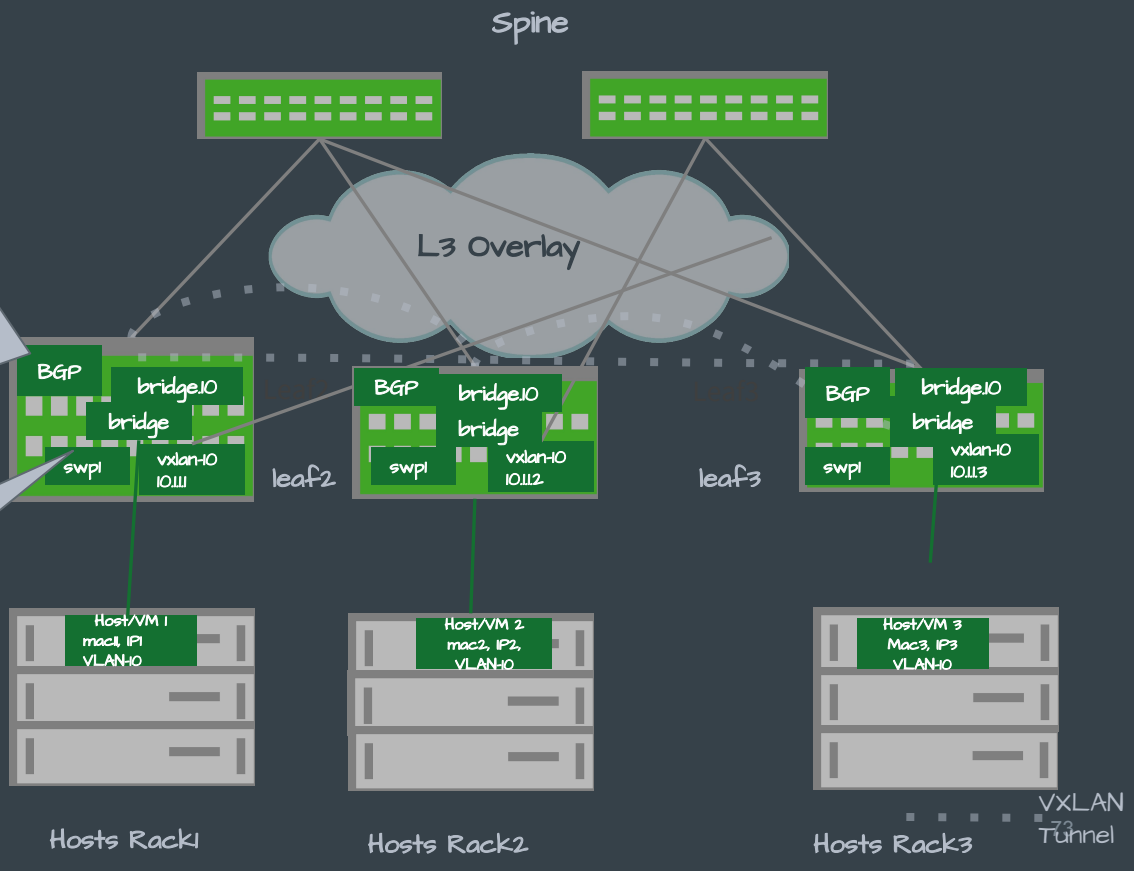
(a) BGP discovers local vlan-vni mapping via netlink

(b) BGP reads local <mac, ip, vlan> entries and distributes them to bgp E-vpn peers

(c) BGP learns remote <mac, ip, vni> entries from E-VPN peers and installs them in the kernel neigh table

(d) Kernel neigh table has all local and remote <mac + ip> for proxying neigh discovery msgs

(a) Local snooper process snoops <mac, ip> on local ports and adds them to the kernel neigh table



# Deploying E-VPN with Linux Bridge

# Deploy Linux bridge with Tunnel vxlan ports

- Deploy Linux bridge with Tunnel vxlan ports as described previously in the tutorial
- Run BGP on each VTEP
- Configure BGP for E-VPN: example FRR config [13]
- Run local snooper process: to snoop local end-point macs and add to the bridge fdb table
- BGP Listens to neigh notifications and distributes local macs
- BGP adds remote macs from peer with NTF_EXT_LEARNED

Following example only covers a vxlan device per VNI.

# Create all your netdevs (iproute2)

```
$ # create bridge device:
$ ip link add type bridge dev bridge
$ # create vxlan netdev:
$ ip link add type vxlan dev vxlan-10
$ # enslave local and remote ports
$ ip link set dev vxlan-10 master bridge
$ ip link set dev swp1 master bridge
```

(see ifupdown2 [12] example in References section [14])

# Create additional netdevs for neigh entries (E-VPN MAC-IP routes)

```
$ # E-VPN MAC-IP entries (neigh entries) are installed per VNI and
$ # hence per vlan. Hence create per vlan bridge entries for MAC-IP
$ # ie. create vlan devices on bridge
$ ip link add type vlan dev bridge.10
$ # create vxlan netdev:
$ ip link add type vxlan dev vxlan-10
$ # enslave local and remote ports
$ ip link set dev vxlan-10 master bridge
$ ip link set dev swp1 master bridge
```

# Configure vlans

```
$ ip link set dev bridge type bridge vlan_filtering 1
$ #configure vlans
$ bridge vlan add vid 10 dev vxlan-10
$ bridge vlan add vid 10 untagged pvid dev vxlan-10
$ bridge vlan add vid 10 dev swp1
$ bridge vlan add vid 10 dev swp1
$ # Default fdb entries for BUM replication are installed
  by BGP
```

# E-VPN specific config

```
$ #turn off learning on tunnel ports (MAC's are learnt by BGP)
$  bridge link set dev vxlan-10 learning off
# turn on neigh suppression on tunnel ports
$ bridge link set dev vxlan-10 neigh_suppress on
$ # you can further turn off flooding completely on tunnel ports
$ # set unknown unicast flood off
$ bridge link set dev vxlan-10 flood off
$ # set multicast flood off
$ bridge link set dev vxlan-10 mcast_flood off
```

# Check Config

```
$ # Check bridge port flags to make sure all required flags are on
$ bridge -d link show dev vxlan-10
```

# Check your kernel vlan, fdb and neigh state

```
$ bridge vlan show
port     vlan ids
vxlan-10  1 PVID Egress Untagged
                 10
swp1     10 PVID Egress Untagged
         10
bridge None
```

```
$ bridge fdb show
mac1 dev swp1 vlan 10 master bridge
mac2 dev vxlan0 vlan 10 master bridge extern_learn
mac2 vxlan0 dst 10.1.1.2 self ext_learn
mac3 dev vxlan0 vlan 10 master bridge extern_learn
mac3 dev vxlan0 dst 10.1.1.3 self extern_learn

$ ip neigh show
IP1 mac1 dev swp1
IP2 mac2 dev vxlan-10
```

# Troubleshooting and Debugging ..

# Most common problems

- Fdb entries missing from kernel due to control plane netlink errors
- Fdb entries overwritten by learn from hardware or dynamic learn by the bridge or vxlan driver in kernel
- End-point mobility problems:
  - remote end-point or tenant system reachable via vxlan may move to a locally connected node
  - Bridge fdb and vxlan fdb must be kept in sync to avoid black hole or incorrect forwarding behavior

# Debugging using iproute2 and perf probes

- Dumping bridge and tunnel fdb tables:
- $bridge fdb show
  - Both bridge and vxlan fdb tables are dumped
  - Vxlan fdb entries are qualified by dev = <vxlan_dev> and flag 'self'
- Monitoring bridge link and fdb events:
- $bridge monitor [link | fdb]
- In recent kernels use bridge perf tracepoints:
  - $perf probe --add bridge:*

# References


[1] Data center networks: https://tools.ietf.org/html/rfc7938#section-4

[2] Data center clos topology: https://tools.ietf.org/html/rfc7938#section-3.2

[3] A Network Virtualization Overlay Solution using EVPN: https://tools.ietf.org/html/draft-ietf-bess-evpn-overlay-08

[4] Address resolution problems in large data centers: https://tools.ietf.org/html/rfc6820

[5] Framework for Layer 2 Virtual Private Networks (L2VPNs) https://tools.ietf.org/html/rfc4664

[6] VPLS rfc : https://tools.ietf.org/html/rfc4762

# References (Continued)


[7] BGP MPLS based E-VPN: https://www.rfc-editor.org/rfc/rfc7432.txt

[8] Requirements for E-VPN: https://tools.ietf.org/html/rfc7209

[9] E-VPN ARP and ND proxy: https://tools.ietf.org/html/draft-ietf-bess-evpn-proxy-arp-nd-03

[10] Free range routing (FRR): https://frrouting.org/

[11] E-VPN webinar by Dinesh Dutt: http://go.cumulusnetworks.com/l/32472/2017-09-22/95t27t

[12] ifupdown2: https://github.com/CumulusNetworks/ifupdown2

# [13] BGP Config for switches (FRR implementation)

LEAF switch config

```
router bgp 65456
   bgp router-id  27.0.0.21
   neighbor fabric peer-group
   neighbor fabric remote-as external
   neighbor uplink-1 interface peer-group fabric
  neighbor uplink-2 interface peer-group fabric
  address-family ipv4 unicast
      neighbor fabric activate
      redistribute connected
  address-family l2vpn evpn
      neighbor fabric activate
      advertise-all-vni
```

SPINE switch config

```
router bgp 65535
   bgp router-id  27.0.0.21
   neighbor fabric peer-group
   neighbor fabric remote-as external
   neighbor swp1 interface peer-group fabric
   neighbor swp2 interface peer-group fabric
   address-family ipv4 unicast
      neighbor fabric activate
      redistribute connected
   address-family l2vpn evpn
      neighbor fabric activate
```

# [14] ifupdown2 config for E-VPN on LEAF switches

```
# /etc/network/interfaces
# example shows one vxlan device per vni

auto vxlan-10
iface vxlan-10
        vxlan-id 10
        bridge-access 10
        vxlan-local-tunnelip 10.1.1.1
        bridge-learning off
        bridge-arp-nd-suppress on
        mstpctl-portbpdufilter yes
        mstpctl-bpduguard yes
        mtu 9152
```

```
# /etc/network/interfaces
# vxlan device per vni

auto bridge
iface bridge
        bridge-vlan-aware yes
        bridge-ports vxlan-10 swp1
        bridge-stp on
        bridge-vids 10
        bridge-pvid 1

auto bridge.10
iface bridge.10
```

# Thank you!