



EVPN and VXLAN in Linux

Ondřej Zajíček

LinuxDays 2026

Problem Statement

How to virtualize ethernet?



Use Case 1: Network Service Operator

- Network infrastructure
- Points of Presence (PoPs) in different places
- Customers want ethernet VPNs between their sites



Use Case 2: Datacenter

- Racks full of servers
- Servers full of virtual machines
- Top-of-Rack switches
- Central network
- Groups of virtuals should be connected together



Classical Solution: IEEE 802.1Q VLANs

- Ethernet virtual networks over ethernet infrastructure
- Just a simple tag in ethernet frame
- 12bit VLAN ID, network-wide unique



Problems of VLANs

- Small range (4094 VLANs)
- Must be configured per hop
- No standard control plane
- Broadcasts for reachability
- Runs over ethernet underlay



Problems of Ethernet

- Minimal redundancy mechanisms (STP)
- Limited load balancing (LAG)
- Broadcasts for reachability
- Spanning Tree vs Shortest Path First
- TRILL / IEEE 802.1aq not successful



Better Solution

Use IP underlay instead of ethernet

- Mature SPF routing protocols (OSPF / IS-IS)
- Equal-cost multi-path (ECMP)
- No network-wide broadcast

Meet **VXLAN**



What is VXLAN?

Virtual Extensible LAN

- Ethernet overlay network over IP
- Encapsulates Ethernet frames in UDP
- 24bit VNI (VXLAN Network Identifier)
- Just data plane without signaling
- Compatible with ethernet VLANs
- RFC 7348



How VXLAN works?

- Service over IP network
- Terminated at VXLAN Tunnel End Points (VTEPs)
- Ethernet frame enters ingress VTEP
- Is encapsulated as VXLAN UDP packet
- Assigned to specific VNI (in VXLAN header)
- Send to specific egress VTEP through IP network
- On egress VTEP, dispatched based on VNI
- Decapsulated and leaves as ethernet frame



VXLAN Forwarding Entries

- **Switch forwarding entry:**
MAC(+VID) -> interface(+VID)
- **VTEP ingress entry:**
MAC(+VID) -> Egress VTEP IP+VNI
- **VTEP egress entry:**
VNI+MAC -> interface(+VID)



VXLAN in Linux

- Supported by Linux kernel
- Virtual interface of type vxlan
- Used together with bridge interfaces
- Standard iproute2 tools
- Both bridge and vxlan interface has its own forwarding table



VXLAN in Linux – Setup VTEP

```
# ip link add name sw10 type bridge
# ip link set sw10 up

# ip link add name vxlan10 type vxlan id 10 local $IP \
    dstport 4789
# ip link set vxlan10 master sw10
# ip link set vxlan10 up

# ip link add veth0 master sw10
# ip link set veth0 up
```



Handling BUM Frames

- **B**roadcast, **U**nknown-unicast, **M**ulticast (BUM) frames
- Must be sent to all VTEPs in the same virtual LAN
- Can be done using IP multicast, or by ingress replication



VXLAN in Linux – Setup BUM FDB Entries

```
# bridge fdb append 00:00:00:00:00:00 dev vxlan10 \  
    dst 10.1.2.1 vni 10 self permanent  
# bridge fdb append 00:00:00:00:00:00 dev vxlan10 \  
    dst 10.1.3.1 vni 10 self permanent
```



VXLAN in Linux – Show FDB Entries (vxlan10)

```
# bridge fdb show | grep vxlan10
6a:39:6b:89:09:5a dev vxlan10 master sw10
7a:77:46:83:cc:d7 dev vxlan10 master sw10
3a:a0:0d:3b:c8:6c dev vxlan10 master sw10
6a:fc:4d:c1:ab:5d dev vxlan10 vlan 1 master sw10 permanent
6a:fc:4d:c1:ab:5d dev vxlan10 master sw10 permanent
00:00:00:00:00:00 dev vxlan10 dst 10.1.2.1 self permanent
00:00:00:00:00:00 dev vxlan10 dst 10.1.3.1 self permanent
3a:a0:0d:3b:c8:6c dev vxlan10 dst 10.1.2.1 self
7a:77:46:83:cc:d7 dev vxlan10 dst 10.1.3.1 self
6a:39:6b:89:09:5a dev vxlan10 dst 10.1.3.1 self
```



VXLAN in Linux – Show FDB Entries (veth0)

```
# bridge fdb show | grep veth0
9a:09:13:b0:a8:cf dev veth0 master sw10
0a:9f:ee:f9:87:2c dev veth0 vlan 1 master sw10 permanent
0a:9f:ee:f9:87:2c dev veth0 master sw10 permanent
33:33:00:00:00:01 dev veth0 self permanent
01:00:5e:00:00:01 dev veth0 self permanent
33:33:ff:f9:87:2c dev veth0 self permanent
33:33:00:00:00:02 dev veth0 self permanent
33:33:ff:00:00:00 dev veth0 self permanent
```



VXLAN in Linux – Setup No-Learning VTEP

```
# ip link add name vxlan10 type vxlan id 10 local $IP \  
    dstport 4789 nolearning  
# ip link set vxlan10 master sw10  
# bridge link set dev vxlan10 learning off  
# ip link set vxlan10 up
```



VXLAN in Linux – Setup VLAN-Aware VTEP

```
# ip link add name vxlan10 type vxlan local $IP \  
    dstport 4789 nolearning external  
# ip link set vxlan10 master sw10  
# bridge link set dev vxlan10 learning off  
# bridge link set dev vxlan10 vlan_tunnel on  
# ip link set vxlan10 up  
  
# bridge vlan add dev vxlan10 vid 2  
# bridge vlan add dev vxlan10 vid 2 tunnel_info id 12
```



Need for Control Plane

- Setting up all BUM FDB entries manually is too much work
- Regular MAC FDB entries should also be set manually
- MAC addresses can be added and removed dynamically
- We need a control plane for VXLAN

Meet **EVPN**



What is EVPN?

Ethernet VPN

- BGP-based control plane for ethernet VPN services
- Handles auto-discovery of VTEPs
- Distributes MAC reachability
- Supports VXLAN and other data planes
- RFC 7432 (MPLS) and RFC 8365 (VXLAN)



What is BGP?

Border Gateway Protocol

- The Internet routing protocol
- Highly scalable and redundant
- Supports multiple types of routing information
- RFC 4271 (and many others)



How EVPN Works

- Internal BGP is just a distributed key-value database
- Routers/VTEPs advertise routing records
- These are distributed to other routers/VTEPs
- Several types of EVPN records



EVPN Record Types

- Type 1: Ethernet Auto-Discovery (EAD)
- Type 2: MAC/IP Advertisement
- Type 3: Inclusive Multicast Ethernet Tag (IMET)
- Type 4: Ethernet Segment (ES)
- Type 5: IP Prefix (RFC 9136, for L3 services)



MAC/IP Advertisement

- Local data plane MAC learning (bridge table)
- Local IP-MAC learning (neighbor table)
- Other sources (DHCP database)
- MAC announced as Type 2 EVPN route (with VNI)
- Remote VTEPs install it as MAC entry to appropriate bridge table



ARP/ND Suppression

- VTEPs act as ARP/ND proxies
- Based on IP-MAC bindings from MAC/IP advertisements
- Reduces broadcast/multicast traffic in the network



Inclusive Multicast Ethernet Tag

- Represents BUM destination reachable through VTEP
- Announced for all VLANs in EVPN instance regardless of known MACs
- BUM frames are sent to all destinations known from IMET routes



BIRD Internet Routing Daemon

- Routing daemon supporting BGP, OSPF and other routing protocols
- In recent versions: support for EVPN
- Running on Linux and BSD
- Free software (GPL)
- Developed by CZ.NIC



BIRD: BGP Setup

```
evpn table evpntab;  
  
protocol bgp ibgp1 {  
    evpn {  
        import all;  
        export all;  
    };  
  
    local 10.1.1.1 as 1;  
    neighbor 10.1.2.1 as 1;  
}
```



BIRD: EVPN Setup

```
eth table etab10;

protocol bridge {
    eth { table etab10; export all; };
    bridge device "sw10";
}

protocol evpn evpn10 {
    eth { table etab10; };
    evpn { table evpntab; };

    encapsulation vxlan {
        tunnel device "vxlan10";
    };

    rd 1:10;
    route target (rt, 1, 10);
    vni 10;
}
```

BIRD: Show eth route

```
bird> show route table etab3
```

```
Table etab3:
```

```
00:00:00:00:00:00 mpls 13 unicast [evpn2 15:21:16.392] * (80)
```

```
    via 10.1.4.1 on vxlan3 mpls 40
```

```
        unicast [evpn2 15:21:17.793] (80)
```

```
    via 10.1.2.1 on vxlan3 mpls 23
```

```
ca:34:d6:d6:2b:7d    unicast [bridge2 15:21:04.291] * L (0)
```

```
    dev vxlan3
```

```
9a:a1:39:25:16:01 mpls 13 unicast [evpn2 15:21:16.392] * (80)
```

```
    via 10.1.4.1 on vxlan3 mpls 40
```

```
6a:7b:97:5e:82:7d mpls 13 unicast [evpn2 15:21:17.793] * (80)
```

```
    via 10.1.2.1 on vxlan3 mpls 23
```

BIRD: Show EVPN route

```
bird> show route table evpntab
```

```
Table evpntab:
```

```
evpn mac 1:13 0 ca:34:d6:d6:2b:7d * mpls 13 [evpn2 15:21:04.291] * (120)
evpn imet 1:30 0 10.1.3.1 [ibgp2 15:21:17.429 from 10.1.3.1] * (100) [i]
evpn imet 1:23 0 10.1.2.1 [ibgp1 15:21:17.793 from 10.1.2.1] * (100) [i]
evpn imet 1:22 0 10.1.2.1 [ibgp1 15:21:17.793 from 10.1.2.1] * (100) [i]
evpn mac 1:40 0 9a:a1:39:25:16:01 * unicast
    [ibgp3 15:21:16.392 from 10.1.4.1] * (100/20) [i]
    via 10.1.24.1 on ve2 mpls 40
evpn mac 1:22 0 3a:a0:0d:3b:c8:6c * unicast
    [ibgp1 15:21:17.793 from 10.1.2.1] * (100/20) [i]
    via 10.1.21.2 on ve1 mpls 22
evpn mac 1:12 0 aa:5f:7c:3f:3a:bb * mpls 12 [evpn1 15:21:04.291] * (120)
evpn mac 1:23 0 6a:7b:97:5e:82:7d * unicast
    [ibgp1 15:21:17.793 from 10.1.2.1] * (100/20) [i]
    via 10.1.21.2 on ve1 mpls 23
```



BIRD: Show EVPN route

```
bird> show route evpn mac 1:23 0 6a:7b:97:5e:82:7d * all
```

```
Table evpntab:
```

```
evpn mac 1:23 0 6a:7b:97:5e:82:7d * unicast
```

```
  [ibgp1 05:21:17.793 from 10.1.2.1] * (100/20) [i]
```

```
  via 10.1.21.2 on ve1 mpls 23
```

```
  Type: BGP univ
```

```
  BGP.origin: IGP
```

```
  BGP.as_path:
```

```
  BGP.next_hop: 10.1.2.1
```

```
  BGP.local_pref: 100
```

```
  BGP.ext_community: (rt, 1, 3) (generic, 0x30c0000, 0x8)
```

```
  BGP.mpls_label_stack: 23
```





Questions?

<https://labs.nic.cz/>
<https://bird.nic.cz/>