MDNOG

MDNOG 1
8-9 October 2026, Chișinău

ITCARE

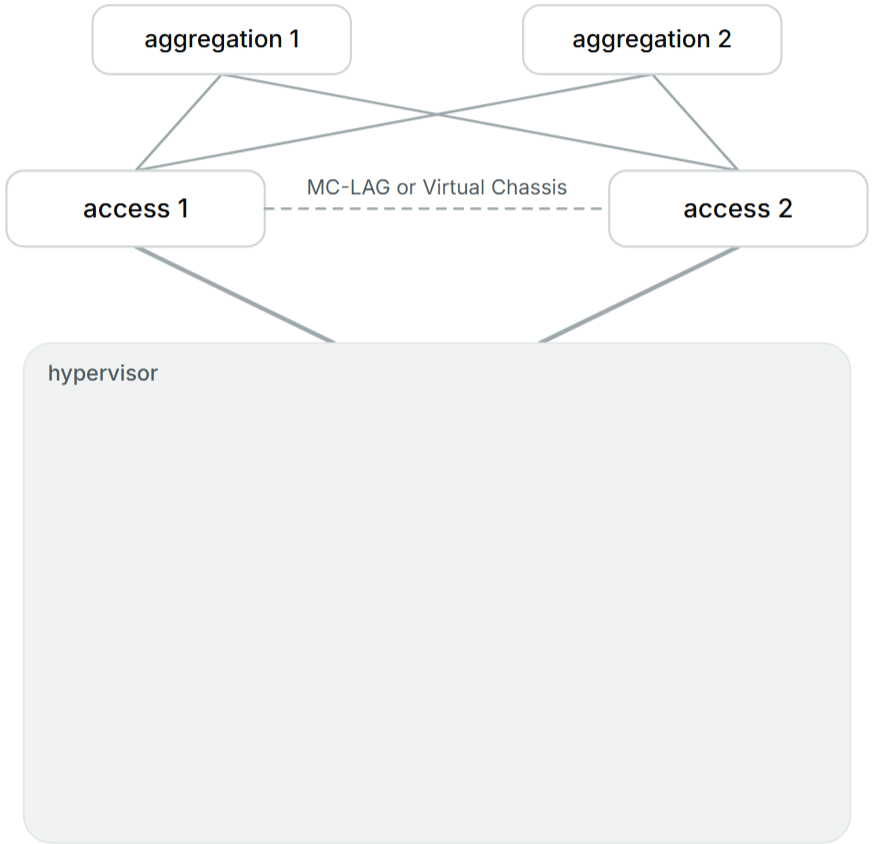
Your leaf is not the edge anymore.

EVPN-VXLAN to the host: route and stretch every workload from the hypervisor.

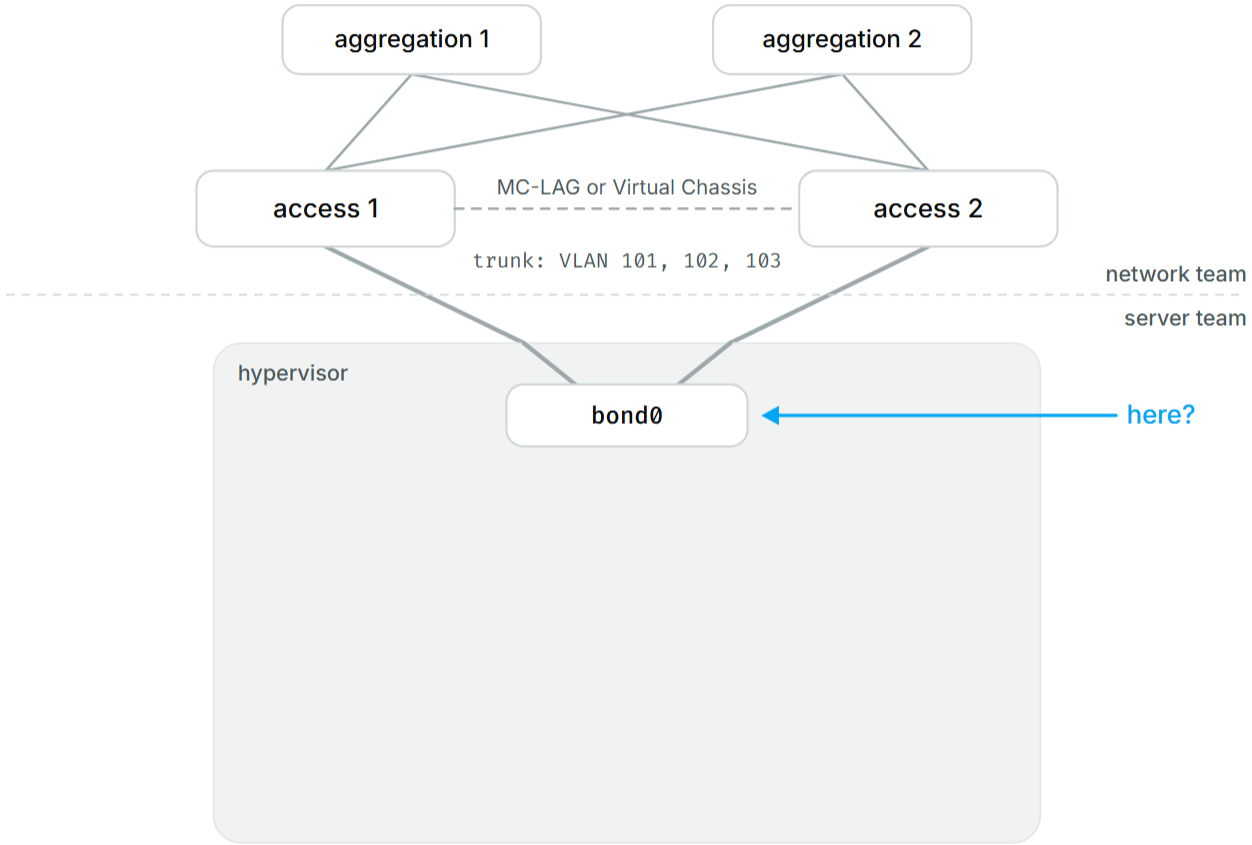
Andrian Visnevschi
CEO & CTO at ITcare

itcare.net

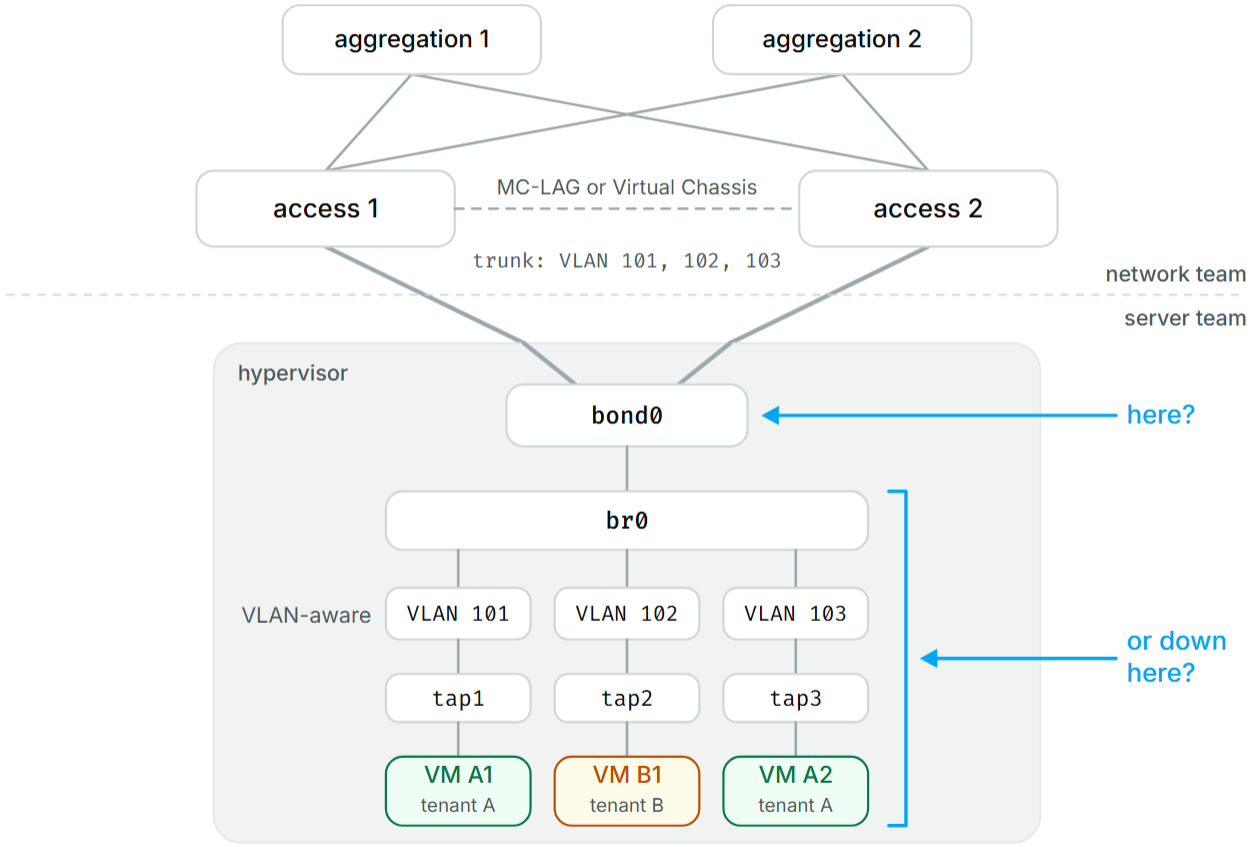
Where does your network end?



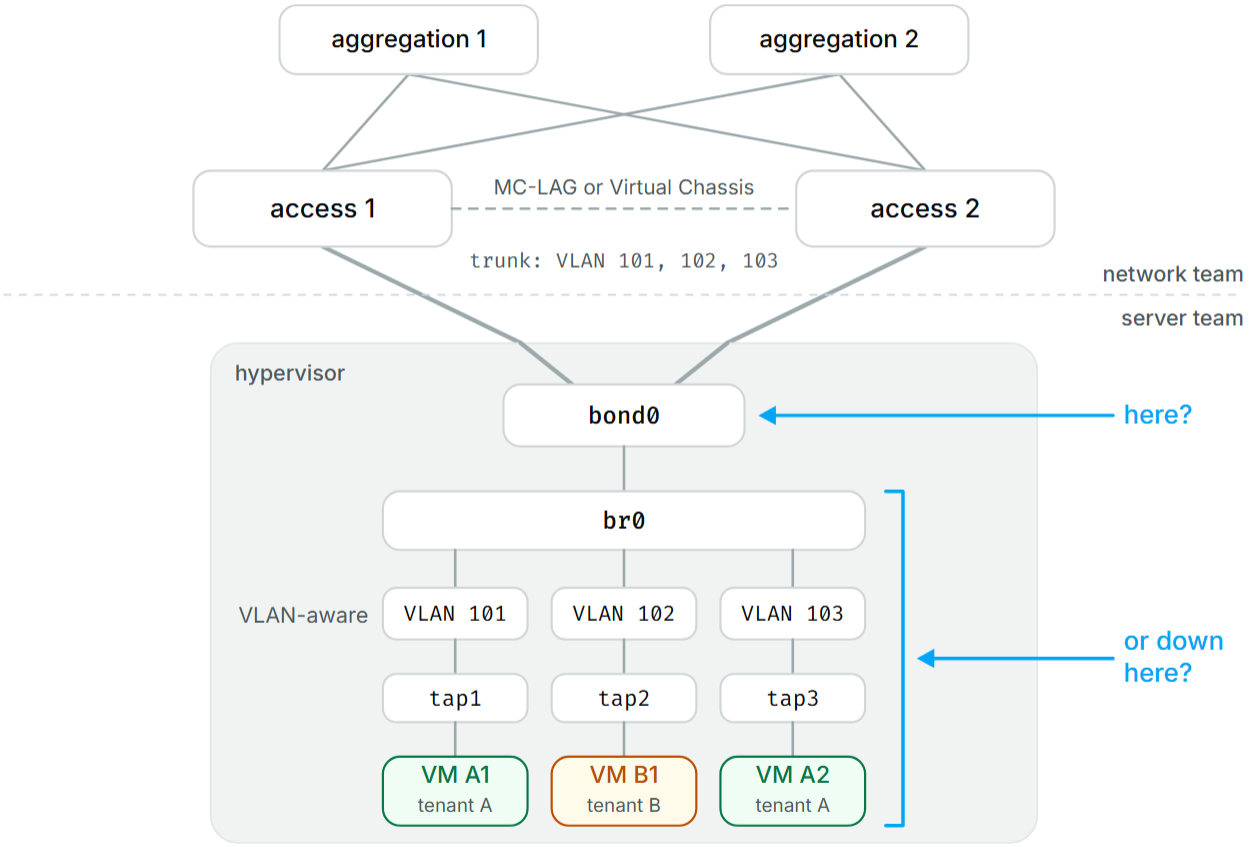
Where does your network end?



Where does your network end?

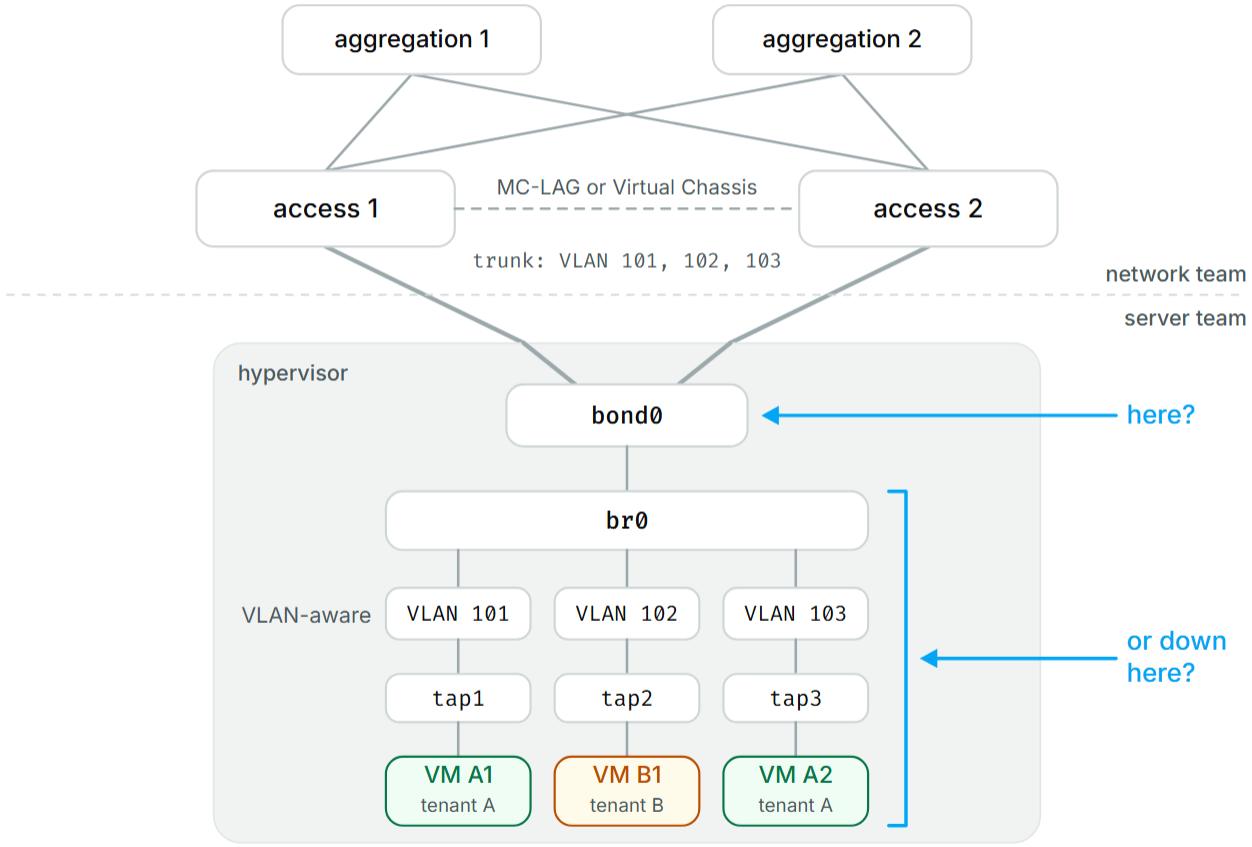


Where does your network end?



Who gets the escalation when something breaks at any of these layers?

Where does your network end?



Who gets the escalation when something breaks at any of these layers?

It's not the network.
There's no way it's the network.
It was the bridge on the hypervisor.

Isn't that part of the network?!

VLAN fabrics: why people stay, and why it bites

Why people still run it



It works on day one, everyone knows it, the switches already do it, and there's nothing new to buy.

Why it bites

- ✕ Spanning tree blocks half your links
- ✕ One broadcast domain is one failure domain: a loop takes out every rack the VLAN reaches
- ✕ Only 4094 VLAN IDs
- ✕ MLAG or Virtual Chassis pairs, with their own failure modes
- ✕ Every small change touches multiple switches

Every rack you add to a VLAN fabric makes the migration harder. That's the argument for starting now.

Your data center, today

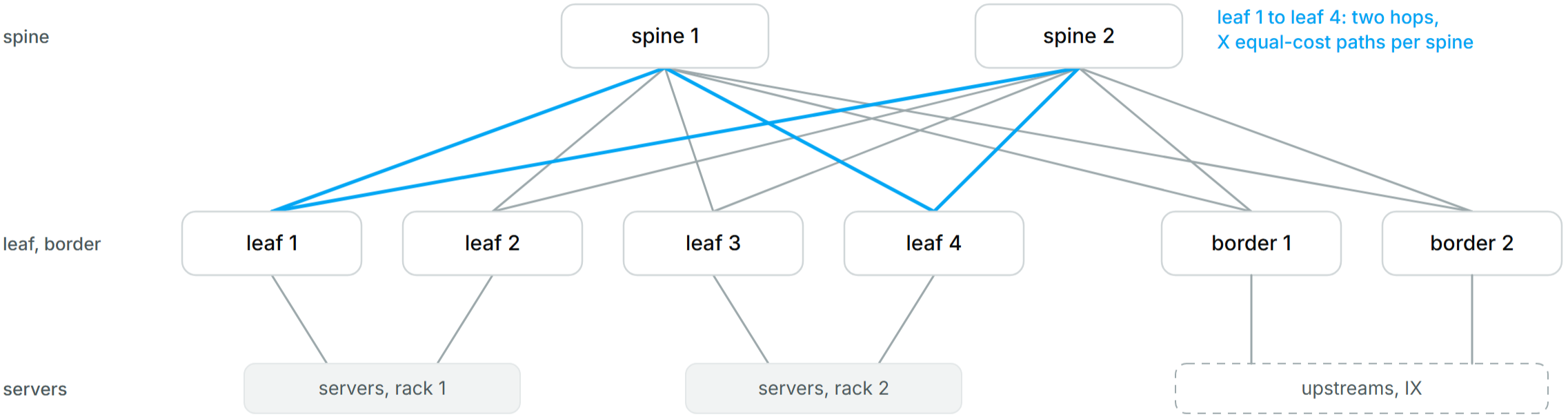
Broadcast storm



Chance	Coverage	Duration
100%	Every rack the VLAN reaches	Until someone unplugs a patch cable

Spanning tree: keeping half your links idle since the last century.

An IP fabric in one picture



Every leaf to every spine

Routed point-to-point links. No spanning tree.

Two hops, always

Any leaf reaches any other leaf through one spine.

Equal-cost paths

X per spine: one per parallel link. ECMP uses them all.

Scale out

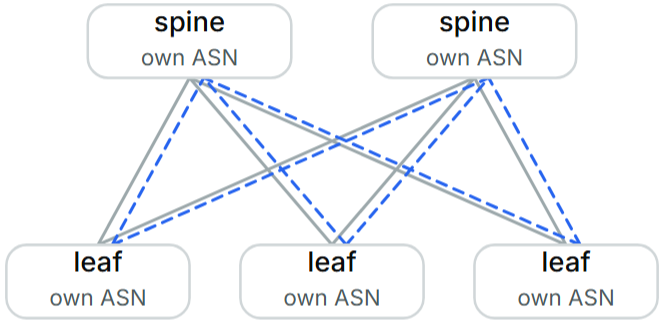
Spines for bandwidth, leaves for ports. Lose a spine: capacity, not reachability.

Reference RFC 7938, Use of BGP for Routing in Large-Scale Data Centers (2016).

Pick your BGP design, and know its caveats

All eBGP

underlay as in RFC 7938, EVPN between loopbacks



links: eBGP dashed: eBGP EVPN, loopbacks

How

An ASN per leaf and per spine; eBGP on the links carries the loopbacks; EVPN runs on separate eBGP sessions between loopbacks.

Good

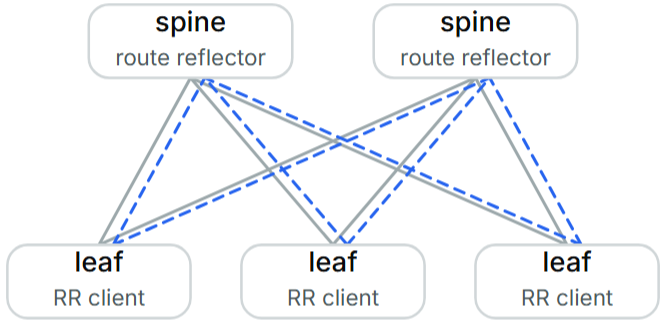
One protocol everywhere, simple policy, the AS path stops loops.

Watch out for

Spines must keep the EVPN next hop unchanged; you need an ASN plan.

eBGP underlay, iBGP EVPN

route reflectors



links: eBGP dashed: iBGP EVPN, loopbacks

How

eBGP on the links for loopbacks; iBGP EVPN from every leaf to route reflectors, often the spines.

Good

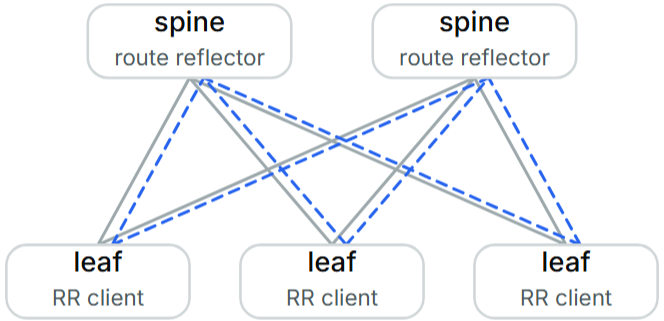
Familiar iBGP, fewer EVPN sessions, next hops untouched by design.

Watch out for

Where the RRs live and what happens when one dies; two BGP worlds to debug.

IGP underlay, iBGP EVPN

OSPF or IS-IS



links: OSPF or IS-IS dashed: iBGP EVPN

How

The IGP carries the loopbacks; iBGP with route reflectors carries EVPN.

Good

Fast convergence, small config.

Watch out for

ECMP paths come from costs, not an AS path: you read the LSDB to find them. Two protocols to run. Inside an area you can't filter what floods into the LSDB; BGP has real route policy.

Plan your ASNs before day one, and make them 32-bit

1,023

private 16-bit ASNs

`64512 to 65534`

94,967,295

private 32-bit ASNs

`4200000000 to 4294967294`

An example scheme: 4200, site, rack, device

spine 1, site 01	`4200 01 00 01`
leaf a, rack 02, site 01	`4200 01 02 01`
every host in rack 02	`4200 01 02 99`

⚠ We learned this the hard way — Our first fabrics are 16-bit. Every new data center is 32-bit, and moving the old ones is the painful part.

There are two kinds of network engineers: those who planned their ASN allocation, and those who are about to.

Sources RFC 6996 (private-use ASNs). RFC 7938, section 5.2.2: the 16-bit range "limited operators to 1023 unique ASNs", hence ASN reuse with allowas-in.

VXLAN and EVPN in two minutes

VXLAN, the data plane

An Ethernet frame inside UDP/IP.

- 24-bit VNI: **16,777,216** segments, not 4094
- Any routed network can carry it
- The VTEP adds and removes the header

EVPN, the control plane

BGP tells every VTEP where each MAC, IP and prefix lives.

- No flood-and-learn across the fabric
- Tenants as VRFs; Layer 2 and Layer 3 in one protocol
- An open standard: several vendors, and FRR

Pros

all links active · no spanning tree · 16 M segments · mobility · multi-vendor

Cons

+50 bytes per packet · new troubleshooting skills · ASIC limits for VTEPs and routes · interop quirks

Flood-and-learn is gossip. EVPN is the meeting minutes.

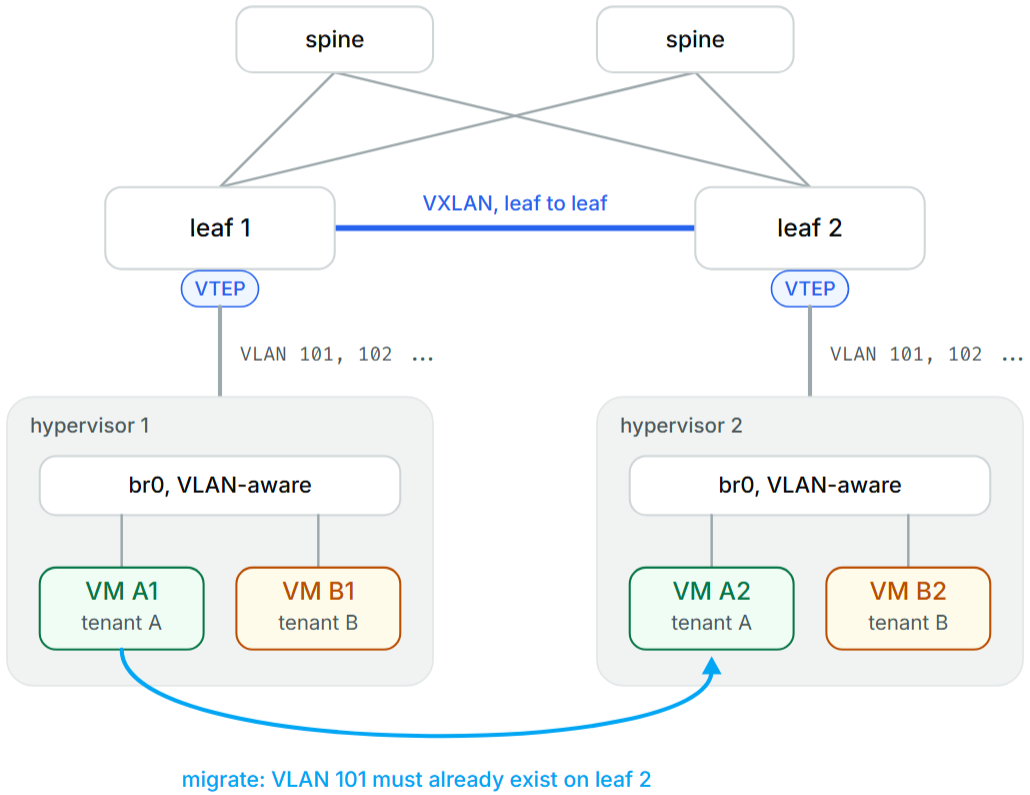
EVPN route types in a classic fabric, leaves as VTEPs

Type	Name	Who sends it	What for
1	Ethernet Auto-Discovery	Leaves with a multihomed (ESI) server link	Multihoming: fast failover and load balancing. Not for finding VTEPs
2	MAC/IP Advertisement	Every leaf, for each MAC and IP behind it	Where a MAC lives; ARP suppression; MAC mobility
3	Inclusive Multicast Ethernet Tag	Every leaf, once per VNI	"I'm a VTEP in this VNI": how VTEPs find each other and build the flood list
4	Ethernet Segment	Leaves sharing an ESI	Find the other leaves on the segment; elect the designated forwarder
5	IP Prefix	Leaves and borders	Routed prefixes in a VRF: subnets, the default route

Move the VTEP to the host and hosts send only types 2, 3 and 5: their uplinks are routed, not multihomed, so they never need types 1 and 4. We still run this classic design too: some rack pairs are dedicated to bare metal, with the leaves as VTEPs.

Sources RFC 7432 (types 1 to 4: sections 8.1, 8.2, 8.4, 8.5, 11), RFC 9136 (type 5), RFC 8365 (EVPN over VXLAN).

Most EVPN fabrics still stop at the leaf



The leaf is the VTEP. The host gets a VLAN trunk, made redundant with MLAG or an ESI bond.

The leaf must know every tenant on every server behind it.

Create or move a VM: change the host and the leaf, and keep them in sync.

Migration: the VLAN goes to every rack the VM might land in.

Every VM's MAC becomes a Type 2 route the whole fabric carries.

The shared segment is the attack surface

One VLAN, several /24s, every tenant

- ✕ ARP spoofing
- ✕ MAC spoofing
- ✕ IPv6 Neighbor Discovery spoofing
- ✕ Rogue Router Advertisements
- ✕ Flooding reaches the whole segment: EVPN cuts BUM traffic, but a big shared L2VNI is still a big flooding domain

And the enforcement point is wrong

The switch sees the host's uplink, not the TAPs behind it.

So anti-spoofing moves into ebtables and nftables on every hypervisor, per virtualisation stack, kept in sync with provisioning by hand, or by hope.



Hardening the shared VLAN is maintaining the wrong design more carefully.

Patient: Doctor, it hurts when I stretch VLANs.

Doctor: Then stop stretching VLANs.

Two production networks, one pattern

A European cloud and hosting provider

VMs and LXC containers on LXD and Proxmox VE hosts. Plain FRR.

A wholesale voice carrier

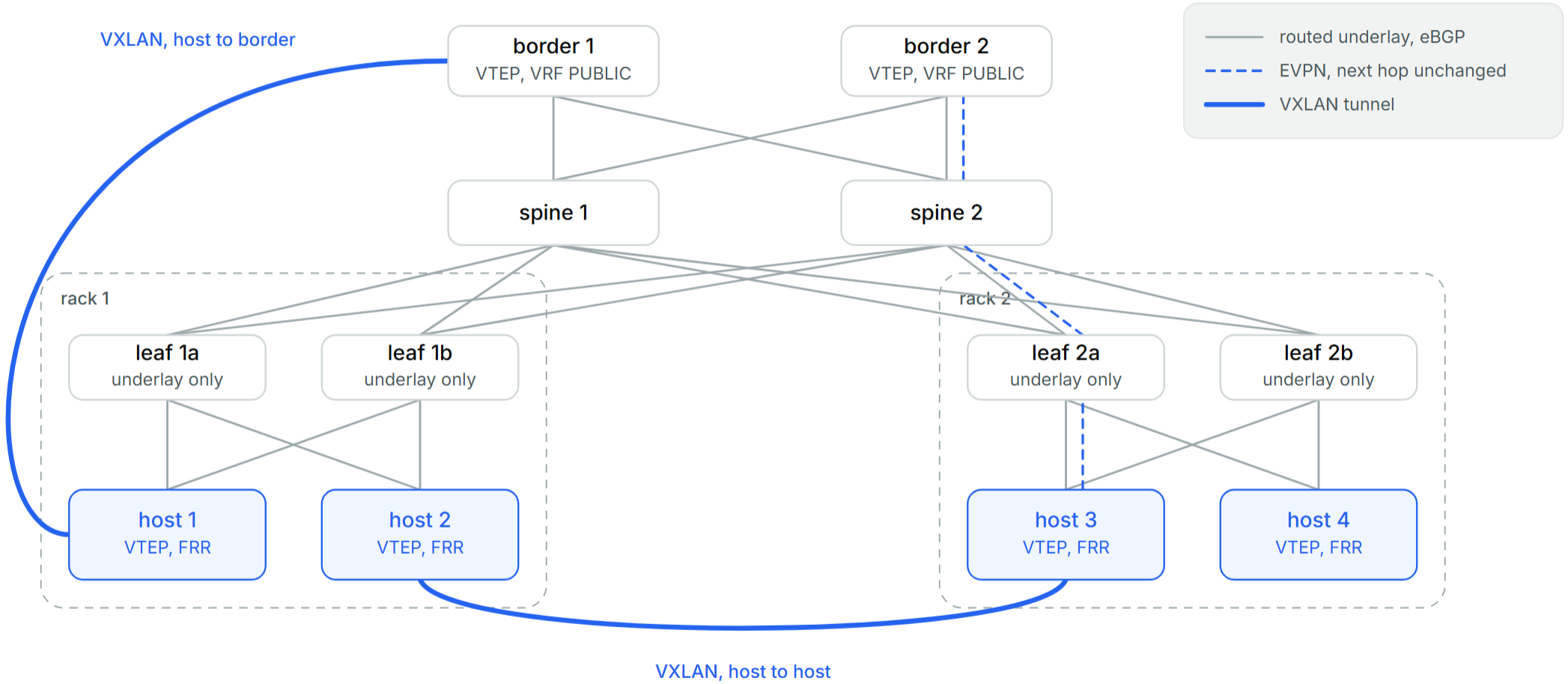
FRR with a VPP dataplane, because voice is lots of small packets.

The pattern

1. The host is the VTEP.
2. L3VNIs and L2VNIs, several VRFs per host.
3. Every workload is a /32 and a /64, sent as EVPN Type 5.
4. For hosts, the leaves only route: no VNI, no tenant VRF, no tenant MAC.

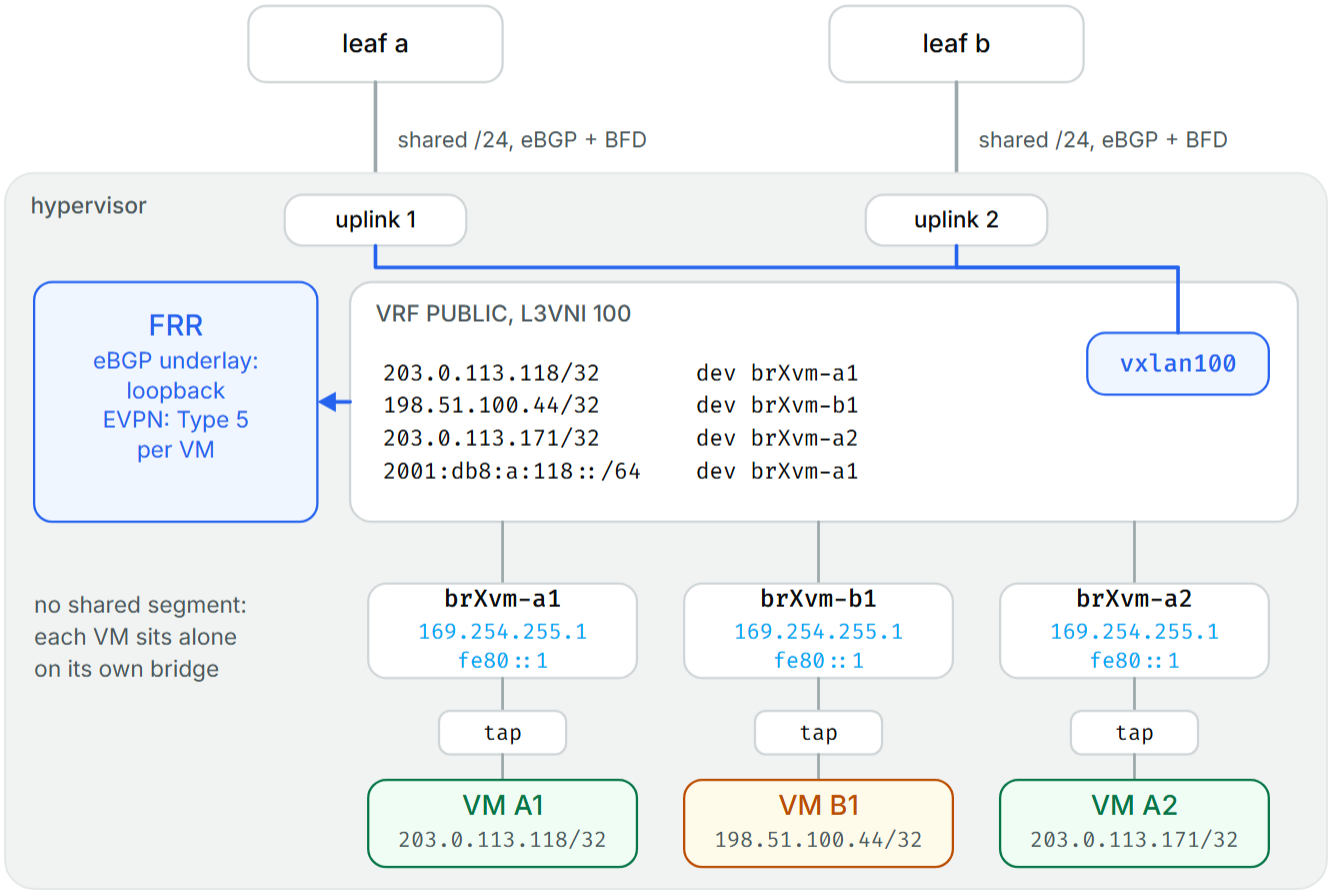
~70	20-30	5	~10,000
instances per host, up to ~250	hosts per rack	racks of leaf pairs	/32s in one VRF

The VTEP lives on the host



For hosts, a leaf has no VNI, no tenant VRF and no tenant MAC. Shown: the hosting provider, where leaves relay EVPN with the next hop unchanged. At the carrier, hosts peer EVPN with route reflectors on the spines.

Inside the host: one bridge per VM, one route per VM



Each VM gets its own bridge, and the bridge sits in the VRF.

Every bridge has the same gateway: 169.254.255.1 and fe80::1.

A static /32 and /64 point at the VM's bridge.

FRR exports them as Type 5 in L3VNI 100. No ARP for a VM ever crosses the fabric.

Link-local: every bridge looks the same

hv-r2-u16

```
root@hv-r2-u16:~# ip link
2: enp36s0f0np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 state UP
4: enp36s0f1np1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 state UP
7: PUBLIC: <NOARP,MASTER,UP,LOWER_UP> mtu 65575 state UP
28: br_public: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master PUBLIC state UP
29: vxlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master br_public state UNKNOWN
22: brXvm-a1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master PUBLIC state UP
35: veXvm-a1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc htb master brXvm-a1 state UP
11: brXvm-b1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master PUBLIC state UP
187: tap5e0c11a2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc htb master brXvm-b1 state UP
```

Hypervisor: the VRF, the L3VNI bridge and two VM bridges

Two VM bridges, two netplan files. Same gateway on both, same VRF; only the name changes.

No /31 per VM, no interconnect IPAM, no overlap. A VM's identity is a route, not an address on a shared subnet.

hv-r2-u16

```
root@hv-r2-u16:~# cat /etc/netplan/brXvm-a1.yaml
network:
    version: 2
    bridges:
        brXvm-a1:
          dhcp4: false
          optional: true
          addresses: [169.254.255.1/24,fe80::1/64]
    vrfs:
      PUBLIC:
        table: 100
        interfaces:
          - brXvm-a1
```

Hypervisor: netplan file of vm-a1's bridge

hv-r2-u16

```
root@hv-r2-u16:~# cat /etc/netplan/brXvm-b1.yaml
network:
    version: 2
    bridges:
        brXvm-b1:
          dhcp4: false
          optional: true
          addresses: [169.254.255.1/24,fe80::1/64]
    vrfs:
      PUBLIC:
        table: 100
        interfaces:
          - brXvm-b1
```

Hypervisor: netplan file of vm-b1's bridge

One cloud-init template, every distribution

vm-a1

```
root@vm-a1:~# cat /etc/netplan/50-cloud-init.yaml
# This file is generated from information provided by the datasource.  Changes
# to it will not persist across an instance reboot.
network:
    ethernets:
        eth0:
            addresses:
            - 203.0.113.118/32
            - 2001:db8:a:118::1/64
            nameservers:
                addresses:
                - 8.8.8.8
                - 1.1.1.1
            optional: true
            routes:
            -   on-link: true
                to: 0.0.0.0/0
                via: 169.254.255.1
            -   on-link: true
                to: ::/0
                via: fe80::1
    version: 2
```

VM running Ubuntu: netplan written by cloud-init

vm-c1

```
[root@vm-c1 ~]# cat /etc/almalinux-release
AlmaLinux release 9.7 (Moss Jungle Cat)
[root@vm-c1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
# Created by cloud-init automatically, do not edit.
#
AUTOCONNECT_PRIORITY=120
BOOTPROTO=none
DEFROUTE=yes
DEVICE=eth0
DNS1=2001:4860:4860::8888
DNS2=2606:4700:4700::1111
GATEWAY=169.254.255.1
IPADDR=203.0.113.140
IPV6ADDR=2001:db8:a:140::1/64
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFAULTGW=fe80::1
IPV6_FORCE_ACCEPT_RA=no
NETMASK=255.255.255.255
ONBOOT=yes
TYPE=Ethernet
USERCTL=no
```

VM running AlmaLinux 9: ifcfg written by cloud-init

Same input, two files. The gateway sits outside the /32, so the route has to be on-link: netplan sets the kernel's onlink flag, while NetworkManager first adds a host route to 169.254.255.1. IPv6 needs neither, because fe80::1 is link-local.

Ubuntu · Debian · AlmaLinux · CentOS · and many others

The whole trick is a few lines of FRR

hv-r2-u16

```
hv-r2-u16# show running-config
vrf PUBLIC
 vni 100
 ip route 203.0.113.118/32 brXvm-a1
 ip route 203.0.113.171/32 brXvm-a2
 ip route 198.51.100.44/32 brXvm-b1
 ipv6 route 2001:db8:a:118::/64 brXvm-a1
 ipv6 route 2001:db8:a:171::/64 brXvm-a2
exit-vrf
!
router bgp 64500 vrf PUBLIC
 address-family ipv4 unicast
  redistribute kernel
  redistribute connected
  redistribute static
 exit-address-family
 !
 address-family l2vpn evpn
  advertise ipv4 unicast
  advertise ipv6 unicast
 exit-address-family
```

FRR on the hypervisor: VRF PUBLIC and its BGP

1. **A VRF with an L3VNI.** vni 100 turns VRF PUBLIC into a routed overlay.
2. **A static route per VM** to its own bridge: a /32 and a /64.
3. **Redistribute static** into BGP inside the VRF.
4. **Advertise as EVPN Type 5.** Every VTEP in the VRF learns it, with this host as the next hop.
5. **ARP does the rest.** The route has no next hop, so the host ARPs for the /32 on that bridge. One bridge, one neighbour: the VM.

hv-r2-u16

```
root@hv-r2-u16:~# arp -an
? (203.0.113.118) at 10:66:6a:00:01:18 [ether] on brXvm-a1
? (203.0.113.171) at 10:66:6a:00:01:71 [ether] on brXvm-a2
? (198.51.100.44) at 10:66:6a:00:02:44 [ether] on brXvm-b1
```

Hypervisor: ARP table, one entry per VM bridge

What the fabric sees

hv-r2-u16

```
root@hv-r2-u16:~# ip route show vrf PUBLIC | head -3
default proto bgp metric 20
        nexthop via 10.0.0.101 dev br_public weight 1 onlink
        nexthop via 10.0.0.102 dev br_public weight 1 onlink
root@hv-r2-u16:~# ip route show vrf PUBLIC | grep brX | head -3
198.51.100.44 dev brXvm-b1 proto static metric 20
203.0.113.118 dev brXvm-a1 proto static metric 20
203.0.113.171 dev brXvm-a2 proto static metric 20
root@hv-r2-u16:~# ip route show vrf PUBLIC | grep -m2 "^[0-9.]* via"
203.0.113.9 via 10.0.1.27 dev br_public proto bgp metric 20 onlink
203.0.113.52 via 10.0.1.41 dev br_public proto bgp metric 20 onlink
```

Hypervisor: routing table of VRF PUBLIC

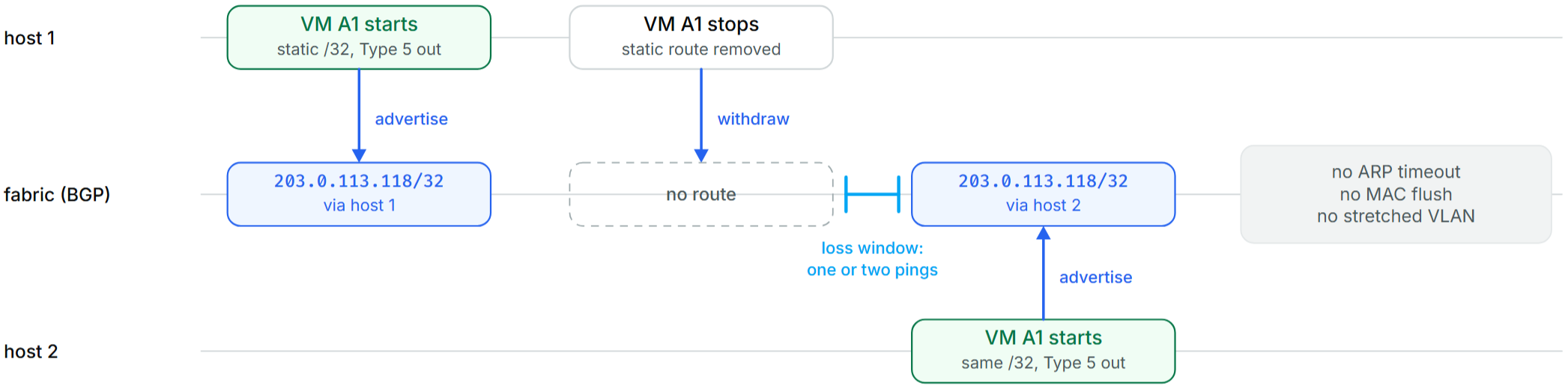
border1

```
noc@border1> show route table PUBLIC.inet 203.0.113.0/24
203.0.113.0/24     *[Static/200] 9w2d 06:43:07
                       Discard
noc@border1> show route table PUBLIC.inet 203.0.113.118 extensive
203.0.113.118/32 (1 entry, 1 announced)
        *EVPN   Preference: 170/-101
                Next hop: 10.0.3.1 via ae4.11
                Next hop: 10.0.3.5 via ae5.21, selected
                Protocol next hop: 10.0.1.13
                State: <Active Int Ext VxlanLocalRT>
                Local AS: 64508
                AS path: 64510 64501 64500 ?
```

Border router: the /24 it announces, and a VM's /32 via its host's VTEP

Local VMs point at their bridge. Remote VMs point straight at another host's VTEP. Everything else goes to the two borders: they carry the full table (about 1.09 million destinations), announce one /24 to the internet, and send each VM's /32 straight to its host's loopback.

IP mobility falls out of the design



Moving a VM is a withdraw and an announce. Convergence is a BGP update, not an ARP timeout.

On an L2VNI, the move is a Type 2 with a higher MAC mobility sequence number (RFC 7432, section 15).

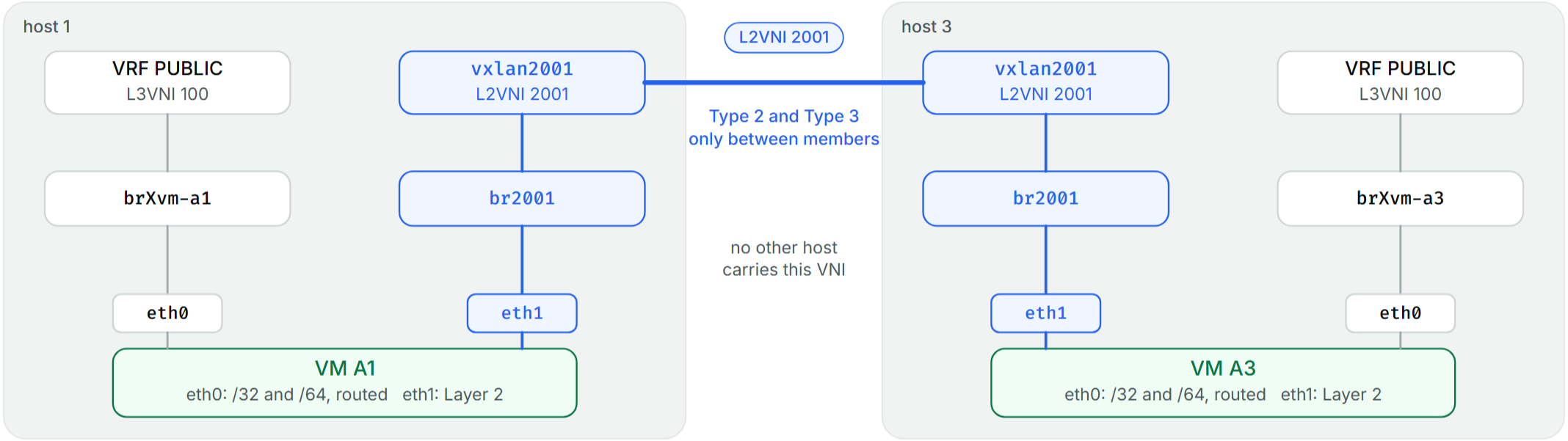
test-client

```
noc@test-client:~$ ping -O -c 6000 -i 0.01 203.0.113.118
...
64 bytes from 203.0.113.118: icmp_seq=3117 ttl=61 time=0.598 ms
no answer yet for icmp_seq=3118
no answer yet for icmp_seq=3119
64 bytes from 203.0.113.118: icmp_seq=3120 ttl=61 time=1.42 ms
...

--- 203.0.113.118 ping statistics ---
6000 packets transmitted, 5998 received, 0.0333333% packet loss, time 59994ms
rtt min/avg/max/mdev = 0.541/0.618/1.420/0.031 ms
```

Test client outside the fabric, 100 pings a second during the move

When a customer needs Layer 2 back



A second interface

eth1 joins an L2VNI with the customer's other VMs only.

Or the same one

On request, eth0 shares an L2VNI instead.

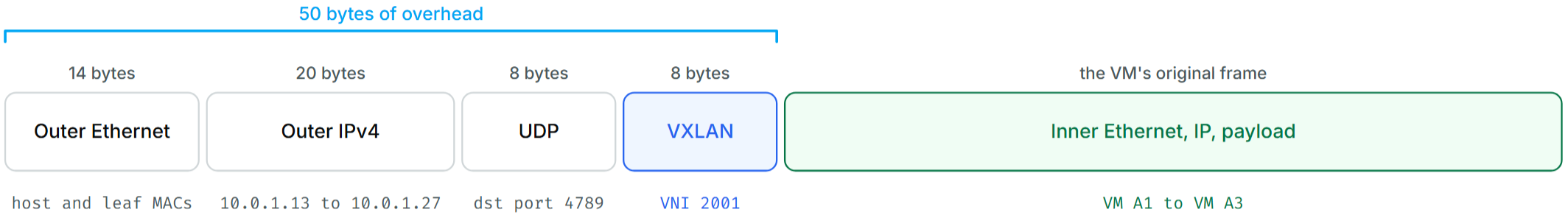
Small blast radius

The VNI exists only on member hosts, and only while a VM needs it.

ARP suppression on

The host answers ARP from what Type 2 already told it, so ARP doesn't flood.

Packet walk: VM A1 to VM A3 on the L2VNI



Outer header: host VTEP loopback to host VTEP loopback. No outer 802.1Q tag; an IPv6 underlay adds 20 bytes.

```
> Frame 1: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits)
> Ethernet II, Src: 0c:42:a1:7e:5d:20, Dst: d4:af:f7:31:9c:04
> Internet Protocol Version 4, Src: 10.0.1.13, Dst: 10.0.1.27
> User Datagram Protocol, Src Port: 52119, Dst Port: 4789
v Virtual eXtensible Local Area Network
  v Flags: 0x0800, VXLAN Network ID (VNI)
      0... .... .... .... = GBP Extension: Not defined
      .... 1... .... .... = VXLAN Network ID (VNI): True
      .... .... .0.. .... = Don't Learn: False
      .... .... .... 0... = Policy Applied: False
      .000 .000 0.00 .000 = Reserved(R): 0x0000
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 2001
    Reserved: 0
> Ethernet II, Src: 10:66:6a:3c:e1:07, Dst: 10:66:6a:91:5b:d2
> Internet Protocol Version 4, Src: 192.0.2.11, Dst: 192.0.2.13
> Internet Control Message Protocol
```

Captured on the hypervisor's uplink, opened in Wireshark

Host uplinks run at MTU 9000, guests at 1500.

Plenty of room for 50 bytes of headers. Raise the MTU on every underlay link.

Miss one, and you'll spend a couple of hours proving it's 100% not the network, it's the application.

And you'll lose that battle.

What a host knows about an L2VNI

hv-r2-u16

```
hv-r2-u16# show evpn mac vni 2001
Number of MACs (local and remote) known for this VNI: 3
Flags: N=sync-neighs, I=local-inactive, P=peer-active, X=peer-proxy
MAC               Type   Flags Intf/Remote ES/VTEP            VLAN  Seq #'s
10:66:6a:91:5b:d2 remote       10.0.1.27                            0/0
10:66:6a:3c:e1:07 local        tapc41f0a97                          0/0
10:66:6a:0e:74:a8 remote       10.0.1.41                            0/0
```

FRR on the hypervisor: MACs in L2VNI 2001

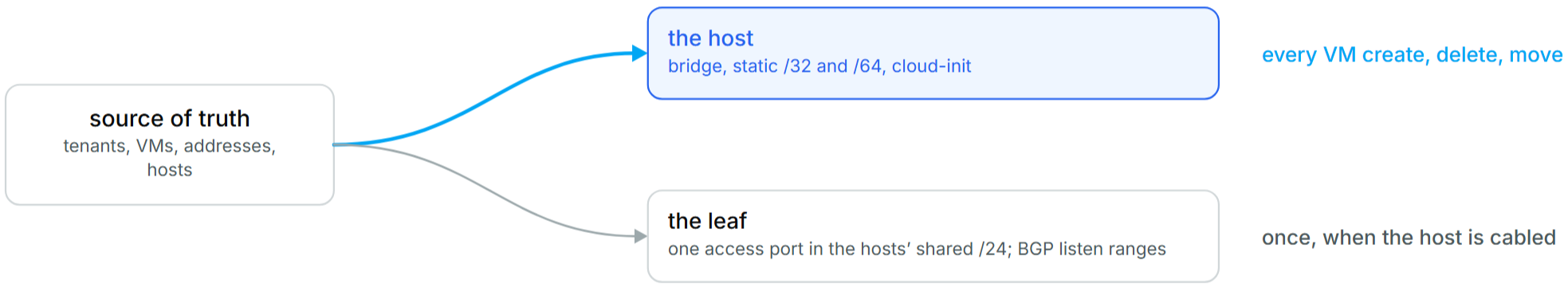
hv-r2-u16

```
root@hv-r2-u16:~# bridge fdb show dev vxlan2001
10:66:6a:0e:74:a8 extern_learn master br2001
10:66:6a:91:5b:d2 extern_learn master br2001
ba:7e:51:0c:93:4d master br2001 permanent
00:00:00:00:00:00 dst 10.0.1.27 self permanent
10:66:6a:91:5b:d2 dst 10.0.1.27 self extern_learn
00:00:00:00:00:00 dst 10.0.1.41 self permanent
10:66:6a:0e:74:a8 dst 10.0.1.41 self extern_learn
```

Hypervisor: the VXLAN device's forwarding table

The all-zero entries are the flood list, built from Type 3 routes. The extern_learn entries came from Type 2: nothing had to be flooded to be found.

What changes in operations



Add a host: cable it.

The leaves accept hosts through BGP listen ranges, so there is no per-host BGP config.

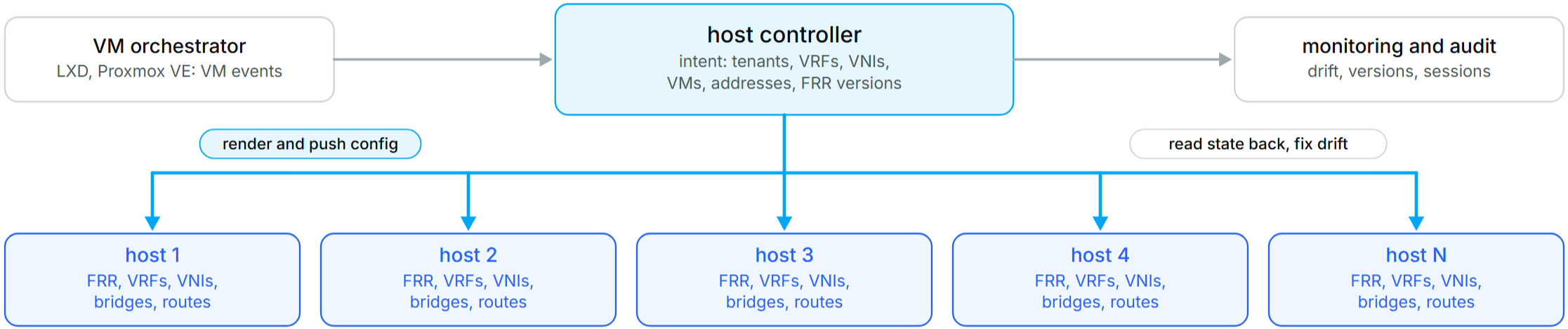
Create, delete or move a VM: the host only.

A bridge, two static routes and a cloud-init file.

Tenants never reach the leaf.

No VNI, VRF or MAC of a VM lives there, so the leaf config doesn't move.

At scale, the hosts need a controller



Host lifecycle

Enrol a host, render its FRR and network config, keep every host on the same FRR version.

Tenants and VNIs

VRFs and L3VNIs where they're needed; L2VNIs spanned only to the hosts whose VMs need them.

Every VM

Bridge, /32 and /64, cloud-init: on create, delete and move, driven by the orchestrator's events.

Housekeeping

Compare intended and actual state; remove stale routes, orphan bridges and empty VNIs.

By hand, this works for ten hosts. Not for a few hundred.

Trade-offs, part 1: complexity moved into the control plane

Every host holds every route in its VRF

About 10,000 /32s in each host's kernel. Linux copes easily; plan for the BGP update rate.

Every host is a BGP speaker

A few hundred hypervisors means a few hundred EVPN peers to filter, and a few hundred FRR installs to keep on one version.

Every host is a VTEP, so BUM floods to each one

Ingress replication sends one copy per VTEP in the VNI. Keep L2VNIs small and turn on ARP suppression.

The borders carry every /32 in hardware

Size the border FIB for the whole VRF, plus growth.

One ASN on every host

A flat leaf config, but allowas-in on every host. RFC 7938, section 5.2.2, explains why that's acceptable.

One bridge per VM, hundreds of interfaces

A bridge and a TAP or veth per VM, so a busy host lists hundreds of links. Linux copes; check that your monitoring and SNMP do too.

Trade-offs, part 2: the host is now a network device

What you take on

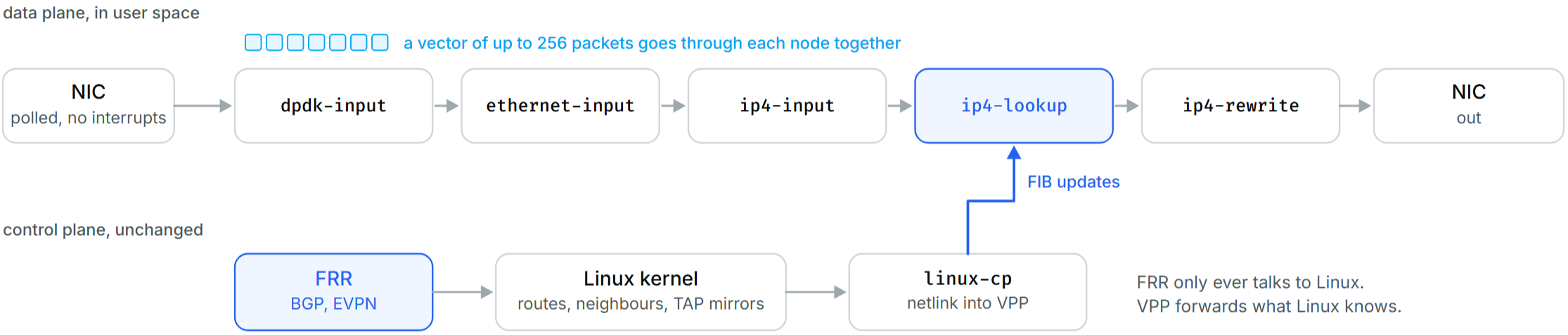
- ⚠ FRR versions, BGP and BFD on every hypervisor: run them like routers
- ⚠ Encapsulation costs CPU: NIC offload, or VPP like the carrier
- ⚠ Underlay MTU raised everywhere: one missed link blackholes quietly
- ⚠ Someone has to own the host network on call

Don't do this if

- ✕ You run a handful of single-tenant servers
- ✕ You have no automation, and no plan for it. At scale, done by hand, it's worse than the shared VLAN.
- ✕ Nobody on the team reads `ip route` and `vtysh`

Congratulations, network engineers: you just joined the server team. You didn't apply.

When packet rate matters: FRR with a VPP dataplane



User space, no interrupts

Poll-mode drivers read the NIC directly. The kernel stack is out of the packet path.

A vector at a time

Up to 256 packets go through each graph node together. The first warms the instruction cache; the rest share the cost.

FRR doesn't change

linux-cp mirrors interfaces into Linux and copies routes and neighbours from netlink into VPP.

What it costs

Cores that poll at 100% even when idle, hugepages, and one more place to debug.

Sources FD.io VPP documentation: scalar vs vector packet processing; the Linux Control Plane (linux-cp) plugin. DPDK programmer's guide: poll mode drivers.

1. The fabric does L3 and only L3.
2. The overlay does tenancy and only tenancy.
3. The automation owns provisioning and only provisioning.

Your leaf is not the edge anymore. The host is.

Route it, automate it, sleep through the night.

Thank you.
Questions?

Andrian Visnevschi
CEO & CTO at ITcare

andrian.visnevschi@itcare.net

itcare.net