
UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

**CLOUD AND DATACENTER NETWORKING - A.A. 2021 - 22**

# *Elaborato:*

# Utilizzo di BGP e E-VPN in una rete leaf-spine di un datacenter simulata tramite Docker containers

Studenti:

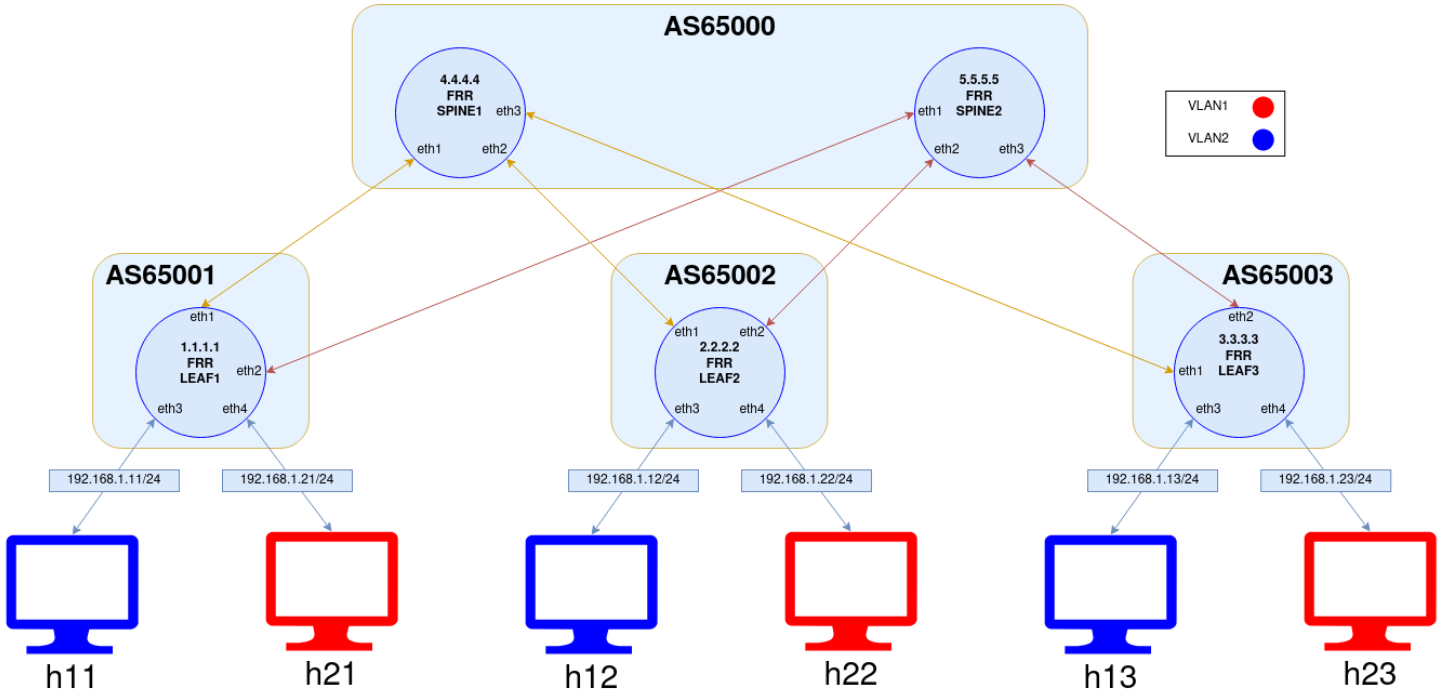
- Luca Campanile M63/1166
  luca.campanile@studenti.unina.it
- Salvatore Criscuolo M63/1143
  salva.criscuolo@studenti.unina.it
- Vincenzo Di Bernardo M63/1374
  vince.dibernardo@studenti.unina.it

*12/10/2023*

# Indice

# 1. Introduzione

L'elaborato proposto descrive le caratteristiche e le strategie di simulazione di una rete **leaf-spine**, configurata per supportare un overlay virtuale. Adottando la tecnica di tunneling **VXLAN**, si consente la comunicazione tra host che sono logicamente separati nell'infrastruttura ma appartengono ad una stessa VLAN, e vengono isolati i flussi di traffico relativi a reti virtuali distinte. L'architettura prevede la divisione dei dispositivi di commutazione in diversi sistemi autonomi (AS), che si scambiano informazioni di routing tramite il protocollo **BGP**.
È previsto, inoltre, l'impiego di ECMP per instradare equamente il traffico verso i percorsi multipli tipicamente presenti in una topologia leaf-spine.
L'implementazione ha richiesto l'utilizzo di **ContainerLab**, strumento che consente la creazione di reti in ambiente virtualizzato mediante containers.



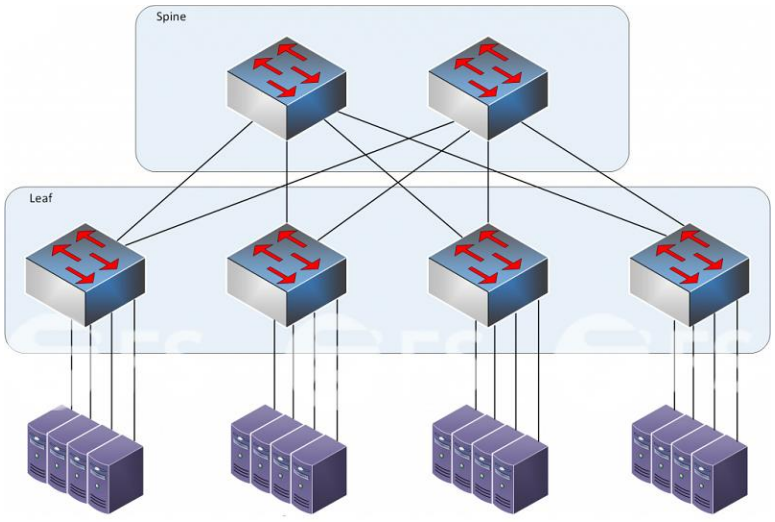
# 2. Topologia Leaf-Spine

## 2.1 Vantaggi architetturali

La rete di un datacenter è tipicamente organizzata in tre livelli:

- **Access Layer**: comprende i dispositivi di commutazione direttamente collegati ai server, in genere appartenenti allo stesso rack;
- **Aggregation Laye**r: è costituito dagli switch che raccolgono e distribuiscono il traffico aggregato proveniente dal livello di accesso;
- **Core Layer**: consente di separare diversi segmenti di rete e garantisce la connessione della struttura con l'esterno.

Una problematica rilevante nella progettazione di un datacenter consiste nell'**oversubscription ratio**, ovvero il rapporto tra la quantità aggregata di traffico trasportato verso un dispositivo e la banda disponibile nel collegamento (uplink) verso un livello superiore. Occorre che il valore di tale metrica sia basso, poiché esso incide sulla probabilità di congestione della rete. Un altro aspetto cruciale è dato dalla robustezza della rete: il guasto di un dispositivo o di un collegamento non deve compromettere la connessione dei server. Per questi motivi, nella realizzazione di un datacenter, è importante adottare soluzioni che introducono **ridondanza**. Oltre alla possibilità di effettuare più collegamenti in parallelo tra un livello e l'altro, risulta conveniente consentire l'instradamento del traffico verso percorsi multipli, in modo che esso possa essere equamente ripartito.  A tale scopo, la topologia di rete **leaf-spine** è quella più comunemente utilizzata nei datacenter.



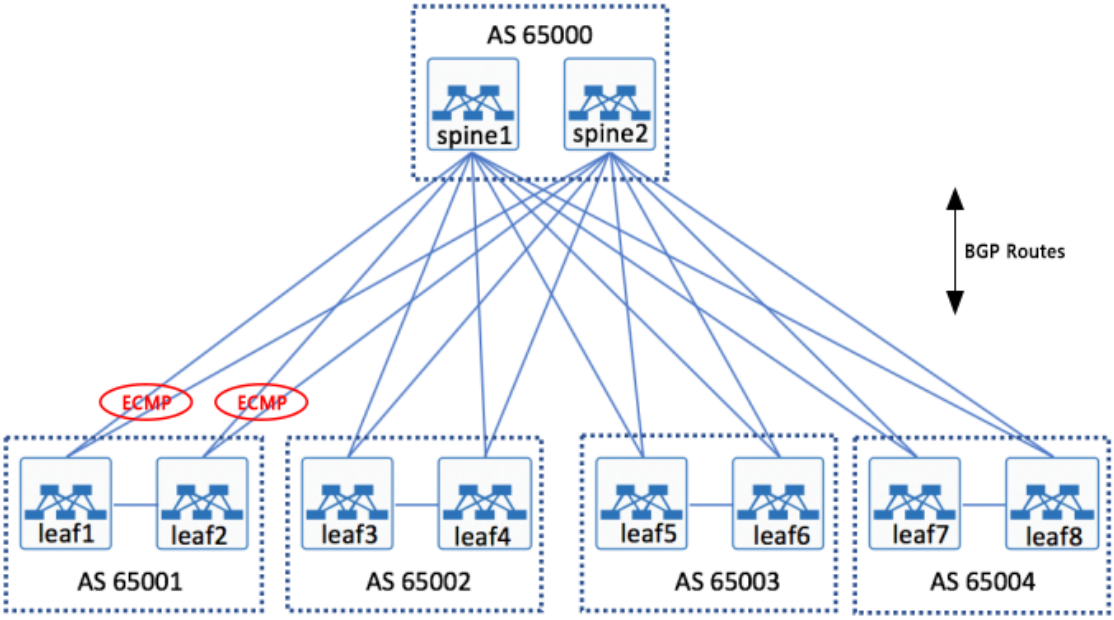
Essa prevede che ogni dispositivo del livello di accesso (leaf) sia collegato con ciascun dispositivo del livello di aggregazione (spine), stabilendo così una struttura altamente interconnessa. Un altro aspetto che rende la realizzazione di una rete leaf-spine particolarmente vantaggiosa è la sua **scalabilità**.
La rete può essere facilmente estesa inserendo, in base alla necessità, altri dispositivi leaf o spine, senza influire sulla connettività esistente.

## 2.2 Strategie di routing

Nel caso in cui i dispositivi di commutazione siano switch in grado di operare esclusivamente al livello 2 dello standard ISO/OSI, è necessario evitare che la ridondanza introdotta nella rete possa causare fenomeni indesiderati come il broadcast storm. Il protocollo STP può prevenire la formazione di loop, ma richiede la disattivazione dei collegamenti che non appartengono ai percorsi previsti dallo spanning tree. Ciò impedisce l'utilizzo di percorsi multipli e il conseguente abbassamento dell'oversubscription ratio, vanificando i vantaggi della topologia leaf-spine. È necessario, quindi, impiegare switch specializzati nell'applicazione di protocolli in grado di gestire efficacemente l'inoltro dei pacchetti, come TRILL o MLPS.

Qualora i dispositivi siano in grado di operare anche al livello 3, è sufficiente adottare opportuni **protocolli di routing** in modo che le rotte per l'instradamento dei pacchetti siano predeterminate e siano evitati i loop.
Se la rete appartiene ad un unico sistema autonomo (AS), il routing interno può essere gestito dal protocollo **OSPF**. Esso prevede lo scambio continuo di pacchetti di tipo link-state tra i dispositivi, in modo da stabilire i percorsi tra ciascuna coppia di nodi della rete, favorendo quelli più brevi o a costo minimo.
Nel caso in cui i dispositivi facciano parte di diversi AS, si ricorre al protocollo **BGP**. I router appartenenti a sistemi adiacenti, si annunciano informazioni sulle rotte rilevate. Tali informazioni includono il prefisso IP che indica una sottorete di destinazione, gli AS già percorsi dall'annuncio, il router di next-hop di una rotta. I router aggiornano le proprie tabelle di instradamento e continuano a scambiarsi messaggi di tipo keep-alive, per rilevare possibili variazioni delle rotte. È possibile configurare i dispositivi in modo che i percorsi tra i nodi della rete siano imposti secondo preferenze.

Il progetto proposto, a scopo dimostrativo, prevede una configurazione in cui ciascun nodo leaf appartiene ad un diverso sistema autonomo, mentre la coppia di nodi spine appartiene allo stesso AS. Le informazioni sui percorsi della rete sono scambiate, appunto, tramite BGP.

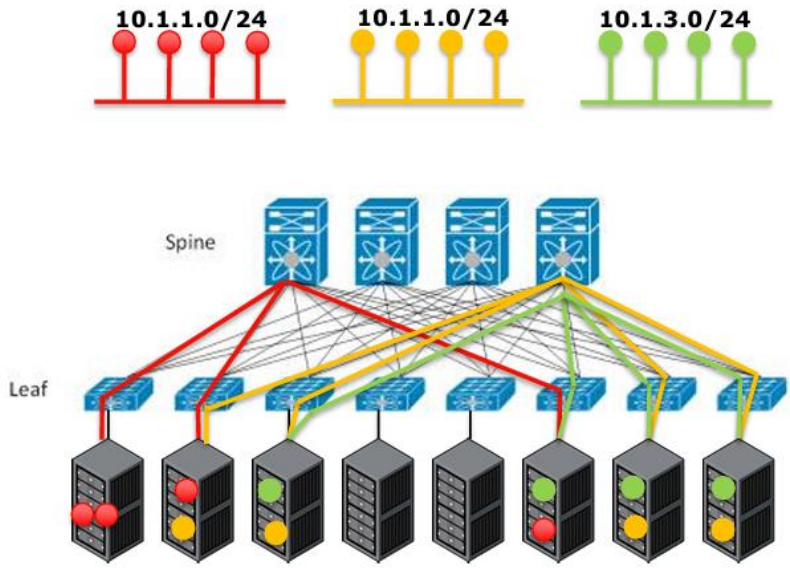



Le strategie descritte prevedono che le scelte di routing si adattino dinamicamente a eventuali cambiamenti o guasti dei collegamenti.
Per distribuire equamente il traffico verso percorsi multipli, la tecnica **ECMP** può essere combinata a BGP (o a OSPF). In corrispondenza di ciascun nodo leaf, la scelta di inoltro di un pacchetto dipende da un algoritmo di bilanciamento che può essere casuale o hash based. Nel secondo caso, l'algoritmo determina lo stesso instradamento per i pacchetti relativi alla stessa connessione TCP, evitandone la possibile degradazione con l'invio di messaggi non in ordine.
Per distinguere connessioni TCP temporalmente disgiunte, è anche possibile adattare ECMP al riconoscimento di differenti flussi TCP (flowlet).

# 3. Overlay Network

## 3.1 VLAN

La tecnologia **VLAN** consente di segmentare la rete di un datacenter in più reti virtuali. I server (o le VMs) riservati ad una certa utenza possono essere virtualmente connessi e interagire in quanto appartenenti ad uno stesso dominio broadcast, pur essendo logicamente separati nell'infrastruttura fisica. Il traffico riservato a distinte reti virtuali risulta isolato anche se gli host interessati sono collegati allo stesso switch di rete.



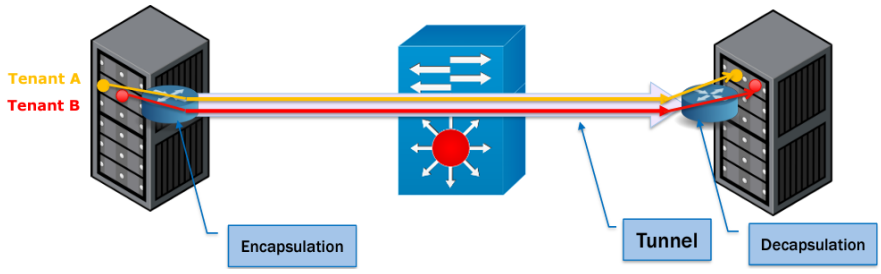

Un amministratore può configurare uno switch assegnando gruppi di porte a diverse VLAN, identificate da un VLAN ID rappresentato su 12 bit. Gli switch conservano e gestiscono autonomamente una forwarding table relativa a ciascuna VLAN. Una rete virtuale può essere estesa su più locazioni di un'infrastruttura collegando gli switch di rete in due modi:

- Tanti collegamenti quante sono le VLAN da estendere. Tale soluzione comporta ulteriori costi di cablaggio e l'occupazione di altrettante coppie di porte dei dispositivi;
- **Trunking**: si effettua un'unica connessione fisica tra i dispositivi e i frame ethernet sono etichettati (**VLAN Tagging**) con un campo che riporta il **VLAN ID**. Tale identificativo è utilizzato per inoltrare il frame verso l'host di destinazione, associato alla VLAN indicata.

La comunicazione tra host appartenenti a VLAN distinte, e quindi a diversi domini broadcast, richiede che l'inoltro sia gestito al livello 3, ovvero tramite un router o uno switch multilayer. Il progetto proposto prevede l'utilizzo di router FRR, configurabili in modo da operare sia al livello 2 che al livello 3.

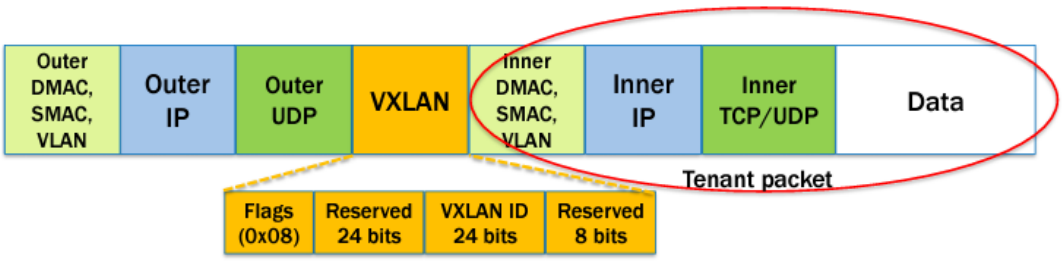
## 3.2 Tunneling e VXLAN

Con l'impiego della tecnologia VLAN, le connessioni tra host appartenenti a diverse reti virtuali sono isolate e si sovrappongo a quelle fisicamente presenti nell'infrastruttura, generando un **overlay**.
Tuttavia, a causa dell'aumento delle utenze e dell'ampio ricorso alla virtualizzazione nei datacenter, soprattutto in quelli destinati ai servizi di cloud computing, il vincolo imposto dai 12 bit sul massimo numero di reti virtuali può risultare restrittivo. Inoltre, un utente che gestisce una struttura di rete nel cloud può avere l'esigenza di adottare una personale segmentazione, stabilendo a sua volta degli identificativi VLAN. Per far fronte a queste problematiche, si utilizzano tecniche di **tunneling**, che prevedono l'incapsulamento dei frame ethernet in pacchetti che includono header speciali, con identificativi dei segmenti virtuali espressi su 24 bit.




In particolare, la tecnologia **VXLAN** prevede che i frame siano incapsulati in pacchetti UDP. I dispositivi che si occupano dell'incapsulamento e decapsulamento dei pacchetti sono chiamati **VTEP**, e possono essere installati a livello software negli switch o nei server della rete.

Un pacchetto VXLAN include:



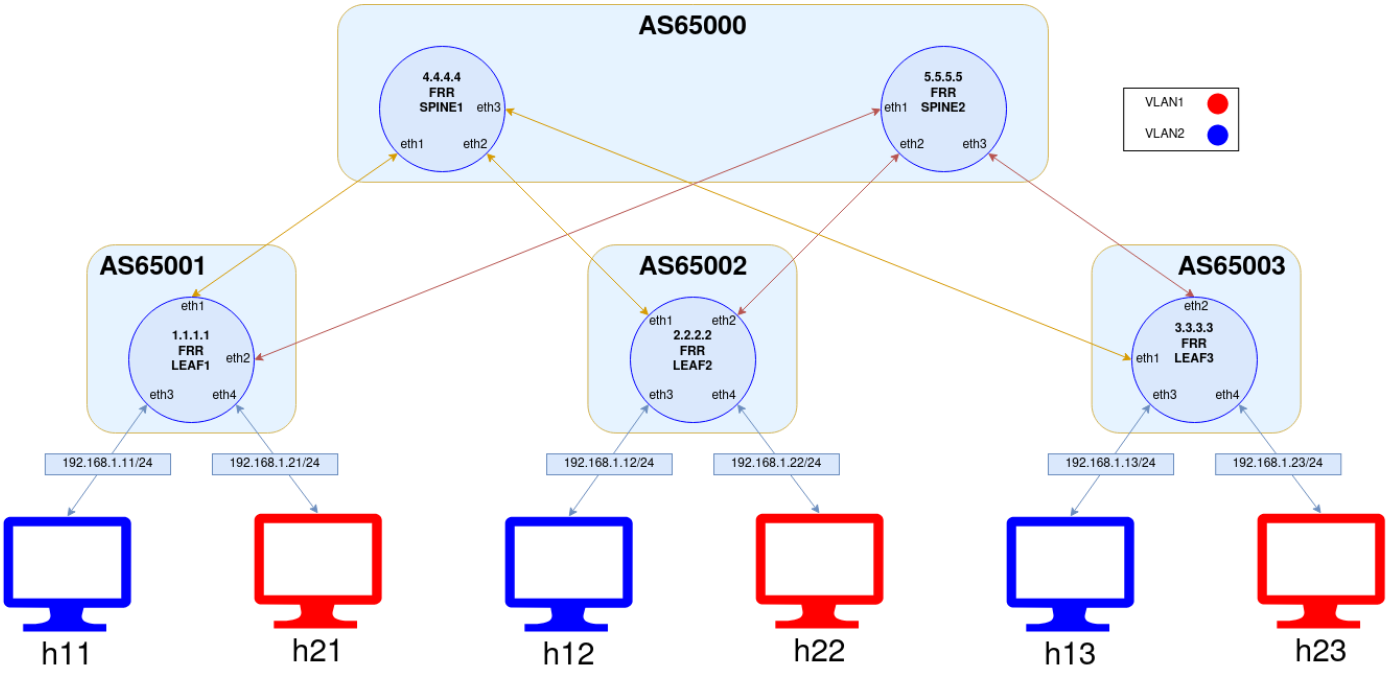

- Header Ethernet: necessario per l'instradamento nell'infrastruttura fisica;
- Header IP: contiene gli indirizzi IP del VTEP sorgente e del VTEP destinazione;
- Header UDP: con numero di porta 4789;
- Header VXLAN: che riporta l'identificativo di un segmento virtuale espresso su 24 bit.
- Payload: frame ethernet originale che può contenere il campo VLAN Tag. Il VLAN ID riportato, in questo caso, può essere riservato all'utenza.

Prima dell'inoltro verso l'uplink, il VTEP sorgente effettua l'incapsulamento ricavando l'indirizzo del VTEP di destinazione a partire dal MAC Address di destinazione presente nel frame e dal VXLAN ID. Il pacchetto ricevuto dal VTEP di destinazione viene decapsulato e inoltrato verso la VLAN e l'host interessati. Un'altra strategia di tunneling nell'ambito delle reti virtuali è **NVGRE**, basata sul protocollo GRE, che prevede l'incapsulamento dei frame in pacchetti IP.

# 4. Implementazione

## 4.1 Configurazione

La simulazione è stata effettuata in ambiente operativo Ubuntu Desktop 22.04.
La rete sviluppata prevede 2 dispositivi spine e 3 dispositivi leaf.
Come previsto dalla topologia, ogni leaf è collegato a ciascuno spine, e viceversa.
Ogni dispositivo leaf appartiene ad un distinto AS, così come la coppia di dispositivi spine. Ad ogni dispositivo è associato un router ID.
Le scelte di routing sono effettuate in seguito allo scambio di rotte BGP.
Ad ogni leaf è connessa una coppia di host, uno appartenente alla VLAN1 e l'altro alla VLAN2. Agli host sono assegnati diversi indirizzi IP che fanno riferimento allo spazio di indirizzamento di sottorete 192.168.1.x/24, scelto per entrambe le VLAN. L'isolamento del traffico è garantito attraverso l'applicazione della tecnologia VXLAN, introdotta configurando opportunamente i dispositivi leaf.




La simulazione prevede la distribuzione di container Docker in ambiente virtuale mediante lo strumento **ContainerLab**. I nodi di commutazione sono router **FRR**, configurabili per supportare diversi protocolli di routing, come IS-IS, OSPF e BGP, e operare sia al livello 2 che al livello 3.
Per ciascun nodo è stato istanziato un container che fa riferimento all'immagine Docker frrouting/frr, e in ciascun container sono caricati file di configurazione inerenti a FRR. Gli host consistono in sistemi Alpine Linux installati come container tramite l'immagine Docker wbitt/network-multitool.

Il tool ContainerLab è in grado di riprodurre la rete desiderata grazie ad un file YAML, che ne riporta la topologia e le specifiche come dati strutturati.

```yaml
name: leaf-spine-vxlan

topology:
  defaults:
    kind: linux
    image: wbitt/network-multitool
  nodes:
    spine1:
      image: frrouting/frr
      binds:
        - config/spine1/daemons:/etc/frr/daemons
        - config/spine1/frr.conf:/etc/frr/frr.conf
    spine2:
      image: frrouting/frr
      binds:
        - config/spine2/daemons:/etc/frr/daemons
        - config/spine2/frr.conf:/etc/frr/frr.conf
    leaf1:
      image: frrouting/frr
      binds:
        - config/leaf1/daemons:/etc/frr/daemons
        - config/leaf1/frr.conf:/etc/frr/frr.conf
        - config/leaf1/setup-vxlan.sh:/etc/frr/setup-vxlan.sh
      exec:
        - ./etc/frr/setup-vxlan.sh
    leaf2:
      image: frrouting/frr
      binds:
        - config/leaf2/daemons:/etc/frr/daemons
        - config/leaf2/frr.conf:/etc/frr/frr.conf
        - config/leaf2/setup-vxlan.sh:/etc/frr/setup-vxlan.sh
      exec:
        - ./etc/frr/setup-vxlan.sh
    leaf3:
      image: frrouting/frr
      binds:
        - config/leaf3/daemons:/etc/frr/daemons
        - config/leaf3/frr.conf:/etc/frr/frr.conf
        - config/leaf3/setup-vxlan.sh:/etc/frr/setup-vxlan.sh
      exec:
        - ./etc/frr/setup-vxlan.sh
    host11:
      exec:
        - ip addr add 192.168.1.11/24 dev eth1 broadcast 192.168.1.255
        - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
    host21:
      exec:
        - ip addr add 192.168.1.21/24 dev eth1 broadcast 192.168.1.255
        - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
    host12:
      exec:
        - ip addr add 192.168.1.12/24 dev eth1 broadcast 192.168.1.255
        - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
    host22:
      exec:
        - ip addr add 192.168.1.22/24 dev eth1 broadcast 192.168.1.255
        - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
    host13:
      exec:
        - ip addr add 192.168.1.13/24 dev eth1 broadcast 192.168.1.255
        - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
    host23:
      exec:
        - ip addr add 192.168.1.23/24 dev eth1 broadcast 192.168.1.255
        - sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
  links:
    - endpoints: ["leaf1:eth1", "spine1:eth1"]
    - endpoints: ["leaf2:eth1", "spine1:eth2"]
    - endpoints: ["leaf3:eth1", "spine1:eth3"]
    - endpoints: ["leaf1:eth2", "spine2:eth1"]
    - endpoints: ["leaf2:eth2", "spine2:eth2"]
    - endpoints: ["leaf3:eth2", "spine2:eth3"]
    - endpoints: ["host11:eth1", "leaf1:eth3"]
    - endpoints: ["host12:eth1", "leaf2:eth3"]
    - endpoints: ["host13:eth1", "leaf3:eth3"]
```

In particolare, sono indicati:

- Per ciascun router:
  - L'immagine Docker frrouting/frr;
  - I file di configurazione FRR daemons e frr.conf da caricare nel container;
  - Nel caso in cui sia un nodo leaf, lo script di shell che consente di annettere al router le due VLAN previste.

- Per ciascun host:
  - L'immagine Docker wbitt/network-multitool;
  - L'esecuzione del comando che consente di assegnare un indirizzo IP statico all'interfaccia eth1 dell'host e 192.168.1.255 come indirizzo broadcast;
  - L'esecuzione del comando che consente all'host di rispondere a pacchetti ICMP (*ping*) di tipo broadcast. Questa opzione è utile per verificare la corretta separazione del traffico delle due VLAN.

- Gli opportuni collegamenti tra le interfacce di host, nodi leaf e nodi spine.

Per ogni router è stata creata una cartella in cui sono presenti i file di configurazione da caricare nei container. Il file di configurazione *daemons* consente di attivare o meno funzioni di routing disponibili in FRR.
Per ogni nodo, in questo caso, è stato abilitato BGP.

```
bgpd=yes
ospfd=no
ospf6d=no
ripd=no
ripngd=no
isisd=no
pimd=no
ldpd=yes
nhrpd=no
eigrpd=no
babeld=no
sharpd=no
staticd=no
pbrd=no
bfdd=no
fabricd=no

vtysh_enable=yes
zebra_options="  -s 90000000 --daemon -A 127.0.0.1"
bgpd_options="   --daemon -A 127.0.0.1"
ospfd_options="  --daemon -A 127.0.0.1"
ospf6d_options=" --daemon -A ::1"
ripd_options="   --daemon -A 127.0.0.1"
ripngd_options=" --daemon -A ::1"
isisd_options="  --daemon -A 127.0.0.1"
pimd_options="   --daemon -A 127.0.0.1"
ldpd_options="   --daemon -A 127.0.0.1"
nhrpd_options="  --daemon -A 127.0.0.1"
eigrpd_options="  --daemon -A 127.0.0.1"
babeld_options="  --daemon -A 127.0.0.1"
sharpd_options="  --daemon -A 127.0.0.1"
staticd_options="  --daemon -A 127.0.0.1"
pbrd_options="   --daemon -A 127.0.0.1"
bfdd_options="   --daemon -A 127.0.0.1"
fabricd_options="  --daemon -A 127.0.0.1"
```

Per i nodi spine è previsto il seguente file di configurazione *frr.conf* :

```
frr version 8.1_git
frr defaults traditional
hostname spine1
no ipv6 forwarding
!
interface lo
 ip address 4.4.4.4/32
exit
!
router bgp 65000
 bgp router-id 4.4.4.4
 bgp log-neighbor-changes
 no bgp ebgp-requires-policy
 timers bgp 3 9
 neighbor LEAF peer-group
 neighbor LEAF advertisement-interval 0
 neighbor eth1 interface peer-group LEAF
 neighbor eth1 remote-as external
 neighbor eth2 interface peer-group LEAF
 neighbor eth2 remote-as external
 neighbor eth3 interface peer-group LEAF
 neighbor eth3 remote-as external
 !
 address-family l2vpn evpn
  neighbor LEAF activate
  neighbor LEAF route-reflector-client
 exit-address-family
 !
 address-family ipv4 unicast
  redistribute connected
 exit-address-family
exit
!
end
```

La configurazione prevede l'assegnazione di un router ID e dell'AS 65000 (uguale per entrambi i nodi spine). È abilitata la registrazione di cambiamenti tramite sessioni BGP con nodi vicini ed è disabilitata la specifica di politiche BGP per la definizione delle rotte. I nodi adiacenti (peer) tengono attive sessioni BGP attraverso lo scambio di pacchetti di tipo keep-alive, che avviene ogni 3 secondi. È previsto un holdtime di 9 secondi, tempo entro il quale, in caso di mancata ricezione di pacchetti keep-alive, una sessione è considerata persa. È definito il gruppo di vicini LEAF e sono indicate le interfacce con cui i nodi spine sono direttamente collegati ai nodi leaf. Ogni nodo vicino comunica gli annunci di rotte BGP in modo immediato. Il termine "external" specifica che i peer appartengono ad altri AS, come previsto dalla topologia adottata.
EVPN è un'estensione di BGP che consente di predisporre la rete alla sovrapposizione (overlay) di traffico relativo alle reti virtuali.

La sezione *address-family l2vpn evpn* consente lo scambio di informazioni sulle reti virtuali tra i nodi leaf (ad esempio, gli indirizzi IP dei VTEP). In tal caso, i nodi spine fungono da "route reflector", ovvero ritrasmettono le informazioni sulle reti virtuali tra un leaf e l'altro tramite le sessioni BGP attive. La sezione *address-family ipv4 unicast* indica che la definizione delle rotte fa riferimento ai soli router ID e che esse vanno annunciate ai nodi direttamente collegati.

Il file di configurazione *frr.conf* relativo ai router leaf presenta poche differenze:

```
frr version 8.1_git
frr defaults traditional
hostname leaf1
no ipv6 forwarding
!
interface lo
 ip address 1.1.1.1/32
exit
!
router bgp 65001
 bgp router-id 1.1.1.1
 bgp log-neighbor-changes
 no bgp ebgp-requires-policy
 timers bgp 3 9
 maximum-paths 2
 neighbor SPINE peer-group
 neighbor SPINE remote-as 65000
 neighbor SPINE advertisement-interval 0
 neighbor eth1 interface peer-group SPINE
 neighbor eth2 interface peer-group SPINE
 !
 address-family l2vpn evpn
  neighbor SPINE activate
  advertise-all-vni
 exit-address-family
 !
 address-family ipv4 unicast
  redistribute connected
 exit-address-family
exit
!
end
```

In questo caso, ogni nodo leaf appartiene ad un diverso AS ed è definito il gruppo di vicini SPINE, raggiungibili mediante le interfacce eth1 e eth2.
L'opzione *maximum-paths 2* introduce l'applicazione di ECMP per le scelte di inoltro, indicando 2 possibili percorsi alternativi che coincidono con i collegamenti con i 2 nodi spine disponibili. Tale specifica è utile per il bilanciamento del traffico e per sfruttare la ridondanza della rete.

La sezione *address-family l2vpn evpn* stabilisce che il nodo leaf deve comunicare le informazioni sulle reti virtuali gestite.

Lo script di shell *setup-vxlan.sh* viene eseguito su ciascun nodo leaf in seguito alla creazione dei rispettivi container. Tale script configura i router adattandoli alla gestione di traffico virtuale mediante VXLAN.

```
#!/bin/bash

ip link add br1 type bridge
ip link set dev br1 up
ip link add br2 type bridge
ip link set dev br2 up
ip link add vxlan10 type vxlan id 10 dstport 4789
ip link add vxlan20 type vxlan id 20 dstport 4789
ip link set dev vxlan10 up
ip link set dev vxlan20 up
ip link set vxlan10 master br1
ip link set vxlan20 master br2
ip link set eth3 master br1
ip link set eth4 master br2
```

Sono creati due bridge virtuali (linux bridge) annettendo ad essi le interfacce verso gli host. Ciascun bridge funge da VTEP occupandosi del tunneling in favore delle VLAN vlan10 e vlan20. Quindi, i pacchetti trasmessi nelle reti virtuali sono incapsulati con VXLAN ID 10 e 20.

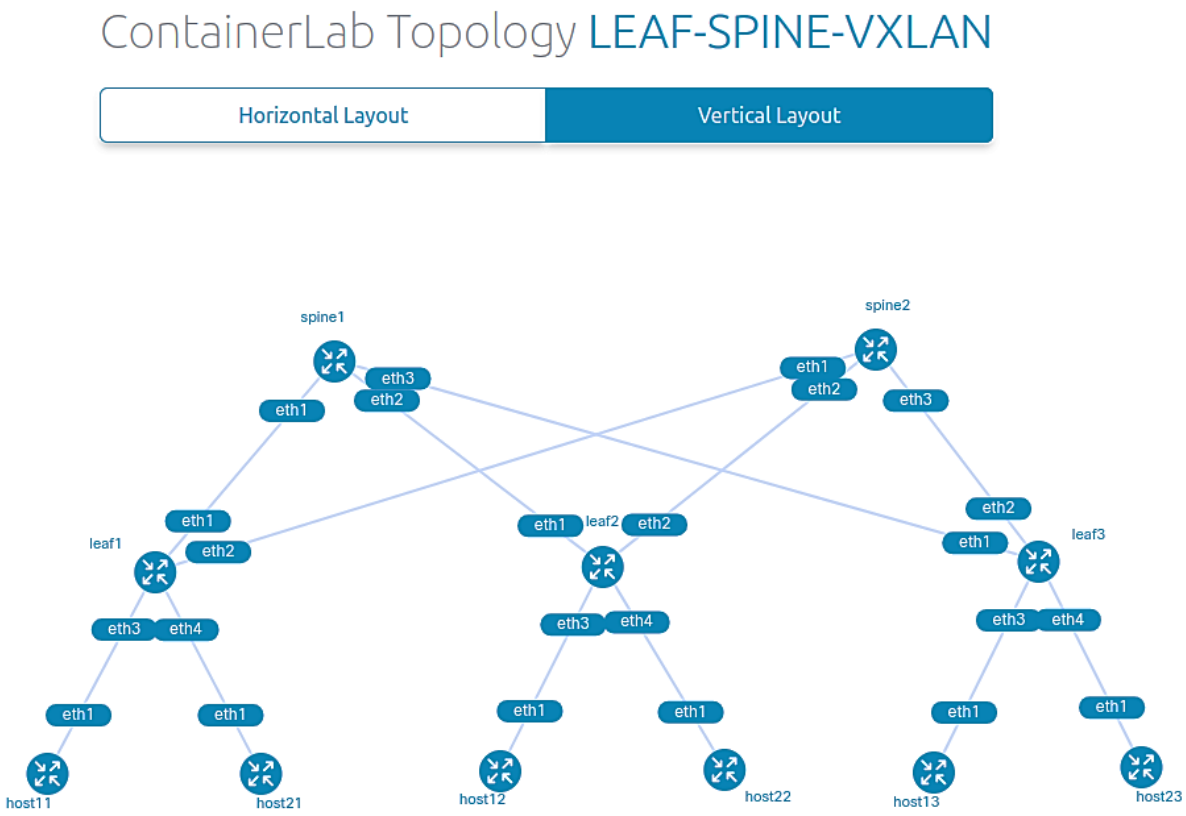
## 4.2 Distribuzione e testing

Avviando il terminale nella cartella di progetto, che contiene tutti i file di configurazione necessari, la creazione dei container in grado di simulare la rete proposta, in accordo con il file YAML, avviene attraverso il comando:

```
sudo containerlab deploy -t leaf-spine-vxlan.yml
```

È possibile ottenere una vista della topologia su un'interfaccia web esposta grazie al seguente comando:

```
sudo containerlab graph -t leaf-spine-vxlan.yml
```




Per verificare la corretta ricezione delle rotte BGP da parte di un nodo leaf, è possibile avviare una shell interattiva nel rispettivo container, avviare lo strumento di configurazione di FRR *vtysh* ed eseguire il comando che consente di visualizzare le rotte scambiate attraverso le sessioni BGP.

```
cruel@cruel-Inspiron-5570:~/LEAF-SPINE-VxLAN_v3$ docker exec -it clab-leaf-spine-vxlan-leaf1 vtysh
% Can't open configuration file /etc/frr/vtysh.conf due to 'No such file or directory'.

Hello, this is FRRouting (version 8.4_git).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

leaf1# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

K>* 0.0.0.0/0 [0/0] via 172.20.20.1, eth0, 00:06:34
C>* 1.1.1.1/32 is directly connected, lo, 00:06:34
B>* 2.2.2.2/32 [20/0] via fe80::a8c1:abff:fe03:42f5, eth2, weight 1, 00:06:30
  *                   via fe80::a8c1:abff:fe2b:fc06, eth1, weight 1, 00:06:30
B>* 3.3.3.3/32 [20/0] via fe80::a8c1:abff:fe03:42f5, eth2, weight 1, 00:06:30
  *                   via fe80::a8c1:abff:fe2b:fc06, eth1, weight 1, 00:06:30
B>* 4.4.4.4/32 [20/0] via fe80::a8c1:abff:fe2b:fc06, eth1, weight 1, 00:06:31
B>* 5.5.5.5/32 [20/0] via fe80::a8c1:abff:fe03:42f5, eth2, weight 1, 00:06:31
C>* 172.20.20.0/24 is directly connected, eth0, 00:06:34
```

Si può notare che in corrispondenza delle rotte verso gli altri nodi leaf sono mostrati più percorsi. ECMP interviene per distribuire il traffico sulla coppia di percorsi multipli. Per verificare l'impiego di ECMP è possibile eseguire, sul leaf1, il comando *ping* prima verso lo spine1 (4.4.4.4), in modo da saturarne il collegamento, e poi verso il leaf3 (3.3.3.3). Avviando lo strumento di analisi del traffico *tshark* sulle interfacce eth1 e eth2 del router, è possibile constatare che lo scambio dei pacchetti ICMP avviene su entrambi i percorsi alternativi, interessando i collegamenti con entrambi i nodi spine.

```
352 111.916974313    1.1.1.1 → 4.4.4.4    ICMP 98 Echo (ping) request  id=0x005a, seq=99/25344, ttl=64
353 111.917015001    4.4.4.4 → 1.1.1.1    ICMP 98 Echo (ping) reply    id=0x005a, seq=99/25344, ttl=64 (request in 352)
354 112.917248696    1.1.1.1 → 4.4.4.4    ICMP 98 Echo (ping) request  id=0x005a, seq=100/25600, ttl=64
355 112.917292977    4.4.4.4 → 1.1.1.1    ICMP 98 Echo (ping) reply    id=0x005a, seq=100/25600, ttl=64 (request in 354)
356 113.917361089    1.1.1.1 → 4.4.4.4    ICMP 98 Echo (ping) request  id=0x005a, seq=101/25856, ttl=64
357 113.917376612    4.4.4.4 → 1.1.1.1    ICMP 98 Echo (ping) reply    id=0x005a, seq=101/25856, ttl=64 (request in 356)
```

```
241 74.052083954    1.1.1.1 → 3.3.3.3    ICMP 98 Echo (ping) request  id=0x005b, seq=60/15360, ttl=64
242 74.052152610    3.3.3.3 → 1.1.1.1    ICMP 98 Echo (ping) reply    id=0x005b, seq=60/15360, ttl=63 (request in 241)
243 75.052300757    1.1.1.1 → 3.3.3.3    ICMP 98 Echo (ping) request  id=0x005b, seq=61/15616, ttl=64
244 75.052367138    3.3.3.3 → 1.1.1.1    ICMP 98 Echo (ping) reply    id=0x005b, seq=61/15616, ttl=63 (request in 243)
245 76.052507277    1.1.1.1 → 3.3.3.3    ICMP 98 Echo (ping) request  id=0x005b, seq=62/15872, ttl=64
246 76.052561558    3.3.3.3 → 1.1.1.1    ICMP 98 Echo (ping) reply    id=0x005b, seq=62/15872, ttl=63 (request in 245)
```

A questo punto, è opportuno controllare la corretta separazione del traffico virtuale tramite i tunnel VXLAN. Mediante il comando *ping*, l'host11 riesce a scambiare pacchetti ICMP con l'host13, appartenente alla stessa VLAN1.

```
cruel@cruel-Inspiron-5570:~/LEAF-SPINE-VxLAN_v3$ docker exec -it clab-leaf-spine-vxlan-host11 sh
/ # ping 192.168.1.13
PING 192.168.1.13 (192.168.1.13) 56(84) bytes of data.
64 bytes from 192.168.1.13: icmp_seq=1 ttl=64 time=0.665 ms
64 bytes from 192.168.1.13: icmp_seq=2 ttl=64 time=0.286 ms
64 bytes from 192.168.1.13: icmp_seq=3 ttl=64 time=0.276 ms
64 bytes from 192.168.1.13: icmp_seq=4 ttl=64 time=0.282 ms
64 bytes from 192.168.1.13: icmp_seq=5 ttl=64 time=0.285 ms
```

D'altra parte, l'host11 non è in grado di rintracciare l'host23, appartenente alla VLAN2.

```
/ # ping 192.168.1.23
PING 192.168.1.23 (192.168.1.23) 56(84) bytes of data.
From 192.168.1.11 icmp_seq=1 Destination Host Unreachable
From 192.168.1.11 icmp_seq=2 Destination Host Unreachable
From 192.168.1.11 icmp_seq=3 Destination Host Unreachable
From 192.168.1.11 icmp_seq=4 Destination Host Unreachable
From 192.168.1.11 icmp_seq=5 Destination Host Unreachable
From 192.168.1.11 icmp_seq=6 Destination Host Unreachable
```

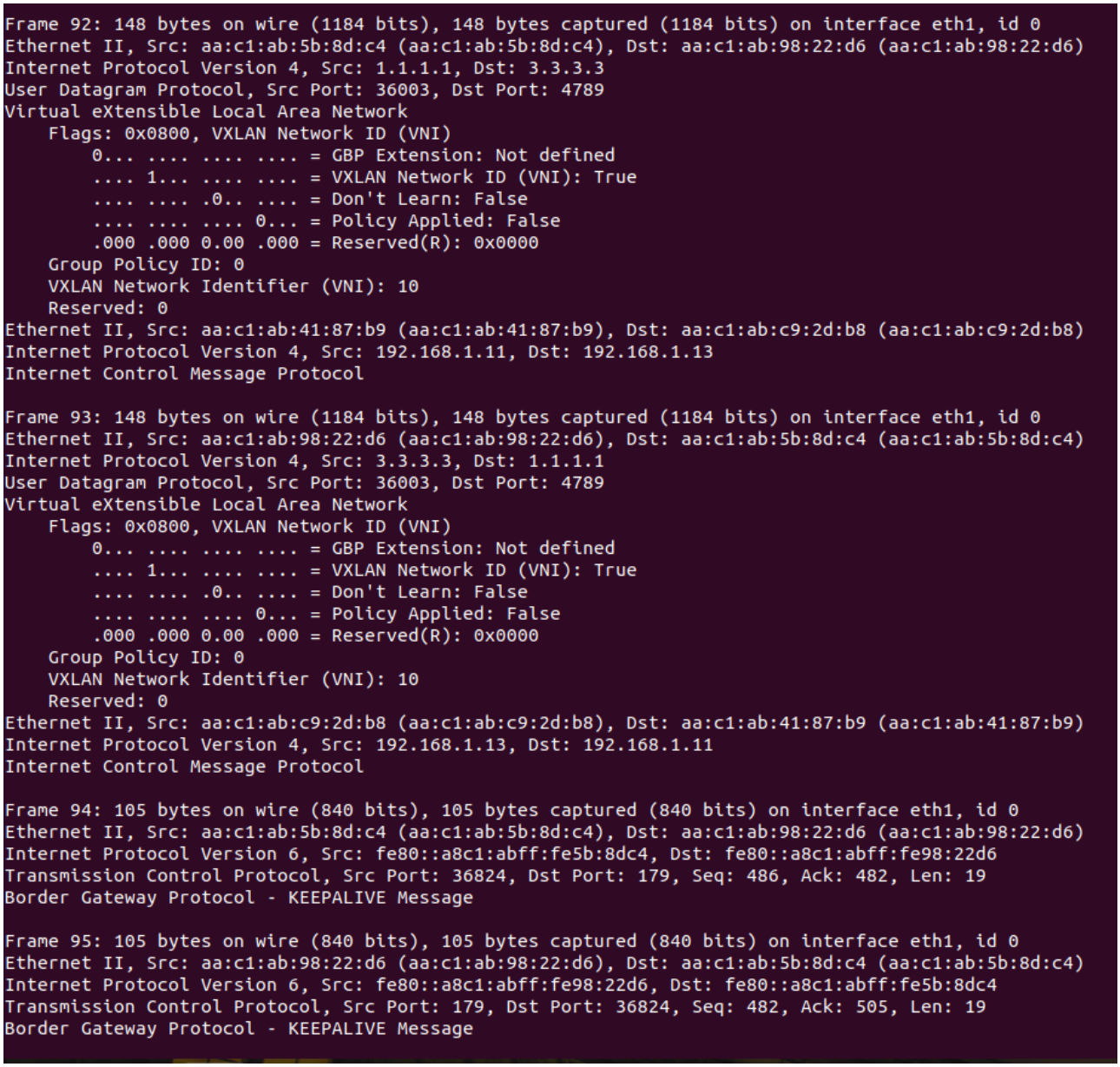
L'host23 è tuttavia raggiungibile dall'host21, appartenente alla stessa VLAN2.

```
cruel@cruel-Inspiron-5570:~/LEAF-SPINE-VxLAN_v3$ docker exec -it clab-leaf-spine-vxlan-host21 sh
/ # ping 192.168.1.23
PING 192.168.1.23 (192.168.1.23) 56(84) bytes of data.
64 bytes from 192.168.1.23: icmp_seq=1 ttl=64 time=0.640 ms
64 bytes from 192.168.1.23: icmp_seq=2 ttl=64 time=0.280 ms
64 bytes from 192.168.1.23: icmp_seq=3 ttl=64 time=0.284 ms
64 bytes from 192.168.1.23: icmp_seq=4 ttl=64 time=0.277 ms
64 bytes from 192.168.1.23: icmp_seq=5 ttl=64 time=0.279 ms
```

Un ulteriore riscontro si può avere inviando pacchetti di *ping* in broadcast verso l'indirizzo 192.168.1.255. Si ricevono pacchetti di risposta solo dagli host appartenenti alla stessa rete virtuale.

```
cruel@cruel-Inspiron-5570:~/LEAF-SPINE-VxLAN_v3$ docker exec -it clab-leaf-spine-vxlan-host11 sh
/ # ping -b 192.168.1.255
WARNING: pinging broadcast address
PING 192.168.1.255 (192.168.1.255) 56(84) bytes of data.
64 bytes from 192.168.1.11: icmp_seq=1 ttl=64 time=0.124 ms
64 bytes from 192.168.1.13: icmp_seq=1 ttl=64 time=0.644 ms
64 bytes from 192.168.1.12: icmp_seq=1 ttl=64 time=0.815 ms
64 bytes from 192.168.1.11: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 192.168.1.13: icmp_seq=2 ttl=64 time=0.584 ms
64 bytes from 192.168.1.12: icmp_seq=2 ttl=64 time=0.592 ms
64 bytes from 192.168.1.11: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 192.168.1.13: icmp_seq=3 ttl=64 time=0.417 ms
64 bytes from 192.168.1.12: icmp_seq=3 ttl=64 time=0.424 ms
```

Infine, facendo riferimento al *ping* tra host11 e host13 e avviando *tshark* sul leaf3, è possibile esaminare il traffico verso l'interfaccia eth1 e constatare lo scambio di pacchetti ICMP incapsulati in datagrammi UDP, in cui sono riportati gli indirizzi IP dei router (e dei VTEP) coinvolti e il corretto VXLAN ID (10).




Dalla schermata si può anche notare l'invio dei pacchetti keep-alive previsti dalla sessione BGP con il nodo spine adiacente.