실습과 함께 완성해보는

도커 없이 컨테이너 만들기

6편

Sam.0

시작하기에 앞서 ...

본 컨텐츠는 앞편을 보았다고 가정하고 준비되었습니다.

원활한 이해 및 실습을 위하여 앞편을 먼저 보시기를 추천드립니다

*네트워크 네임스페이스 **3,4편**에 기초하여 준비되었습니다.*

3편 링크 클릭 *4편 링크 클릭*

실습환경 구성 링크

실습환경

vagrant + virtual vm
ubuntu 18.04, docker (Vagrantfile)
- ubuntu1804 (기존)
- ubuntu1804-2 (추가)

실습 계정 (root)

```
# sudo -Es
```

실습 폴더

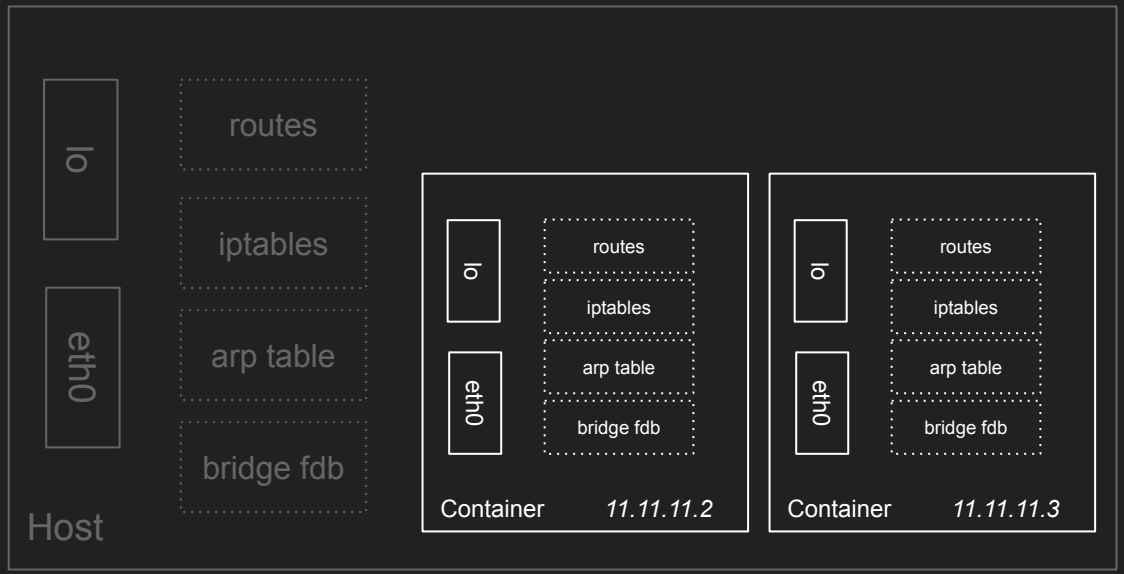
```
# cd /vagrant
```

설치 환경

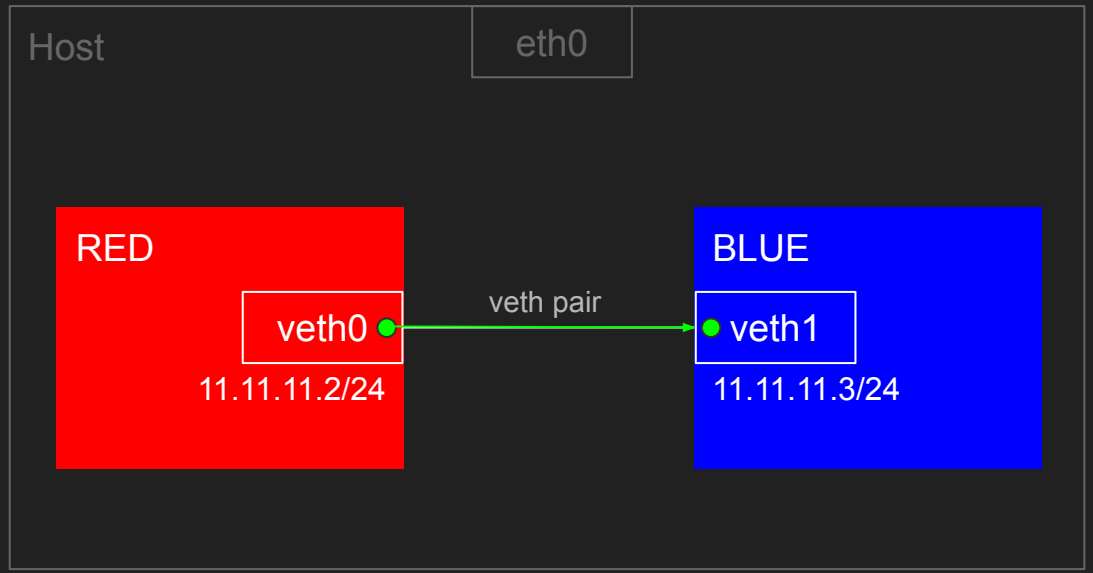
```
# apt-get -y install python-pip > /dev/null 2>&1
# pip install pyroute2
```

Network namespace

호스트 안의 가상 네트워크

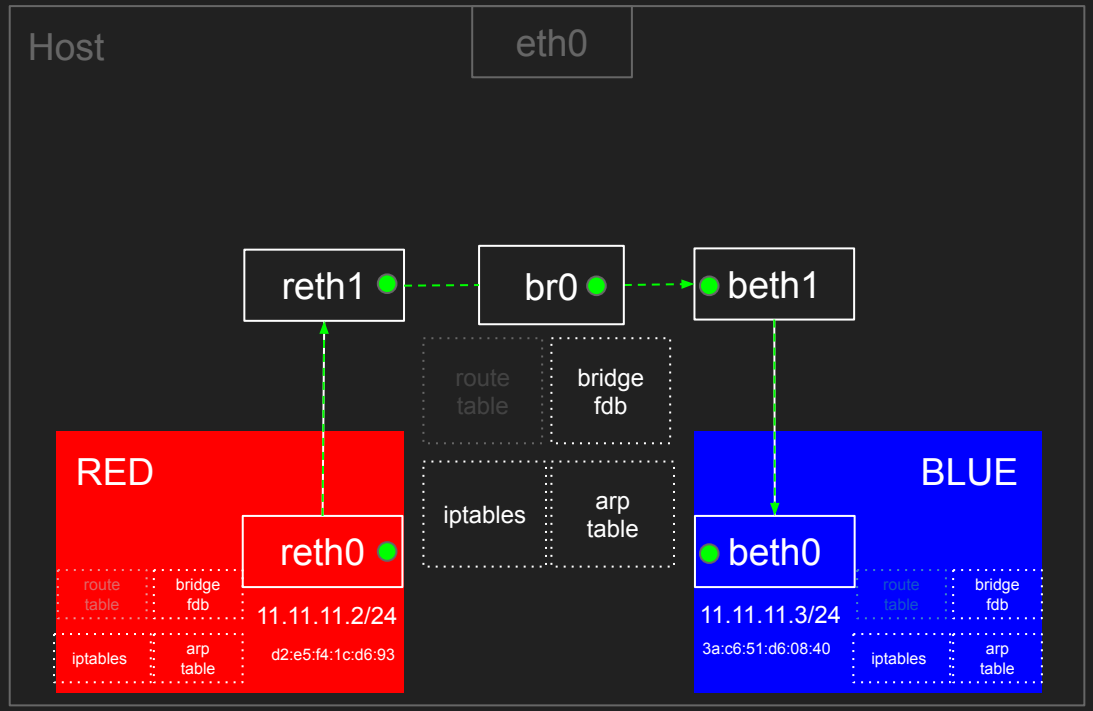


네임스페이스 간 1:1 통신

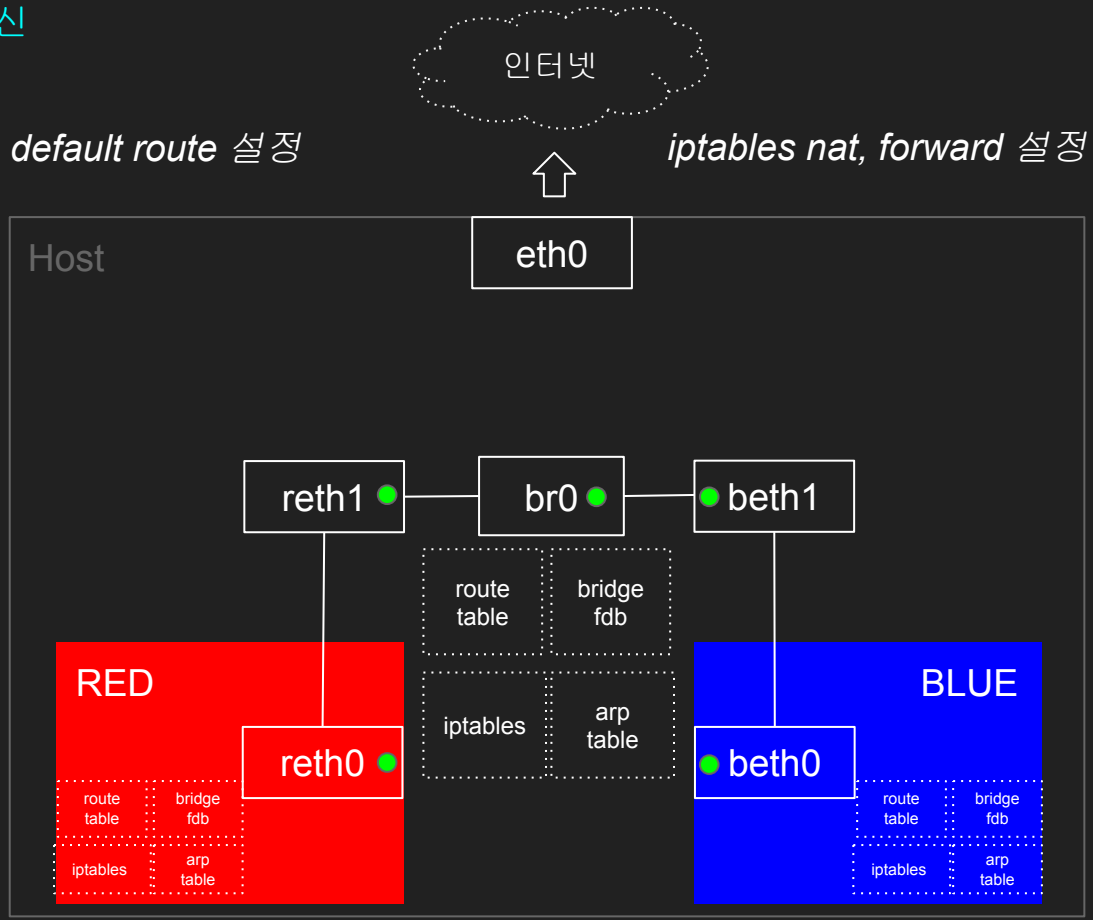


Bridge 통신

arp, fdb정보를 이용하여 통신하고 iptables rule에 따라 전송여부 결정

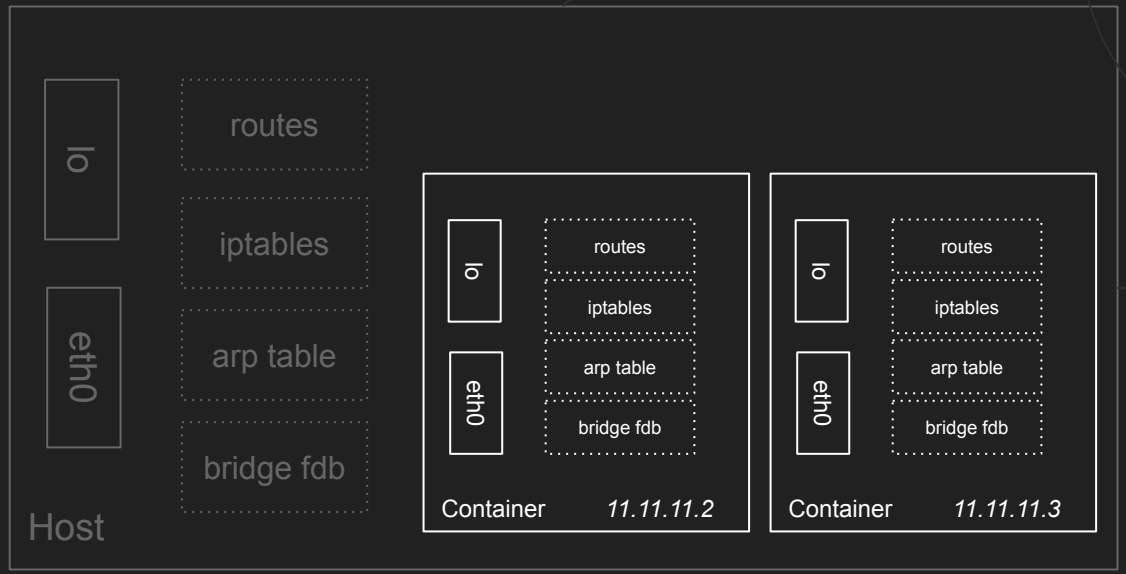


외부 네트워크 통신

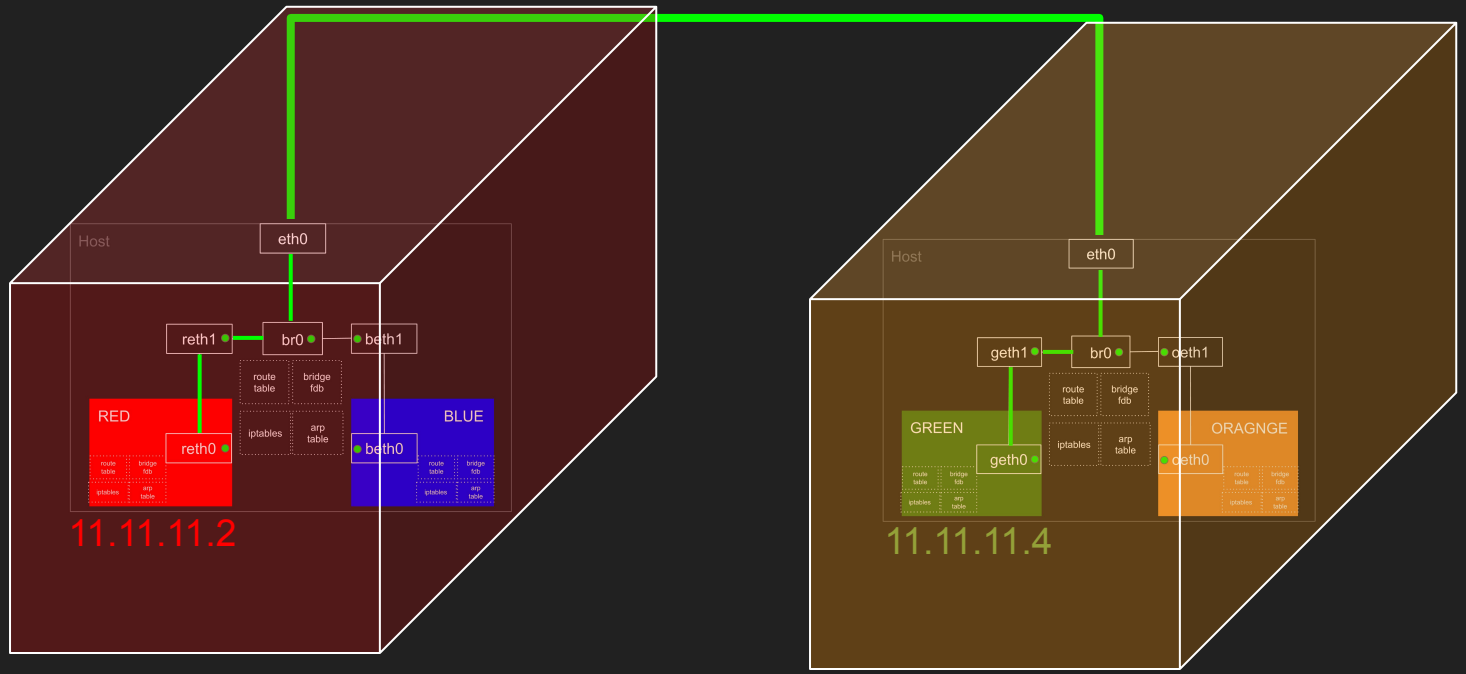

인터넷
default route 설정
iptables nat, forward 설정
Host
eth0
reth1
br0
beth1
route table
bridge fdb
iptables
arp table
RED
reth0
route table
bridge fdb
iptables
arp table
BLUE
beth0
route table
bridge fdb
iptables
arp table

Overlay Network

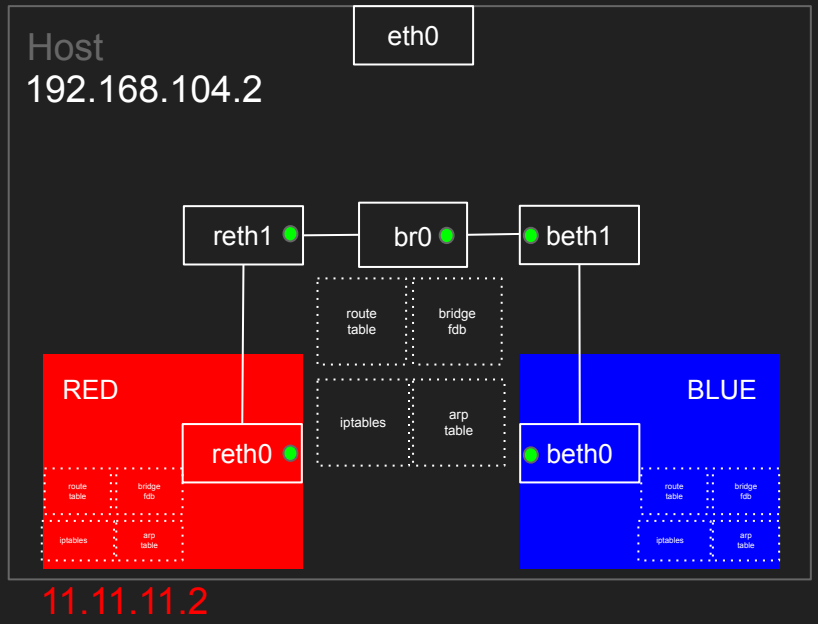
호스트 **간**의 가상 네트워크

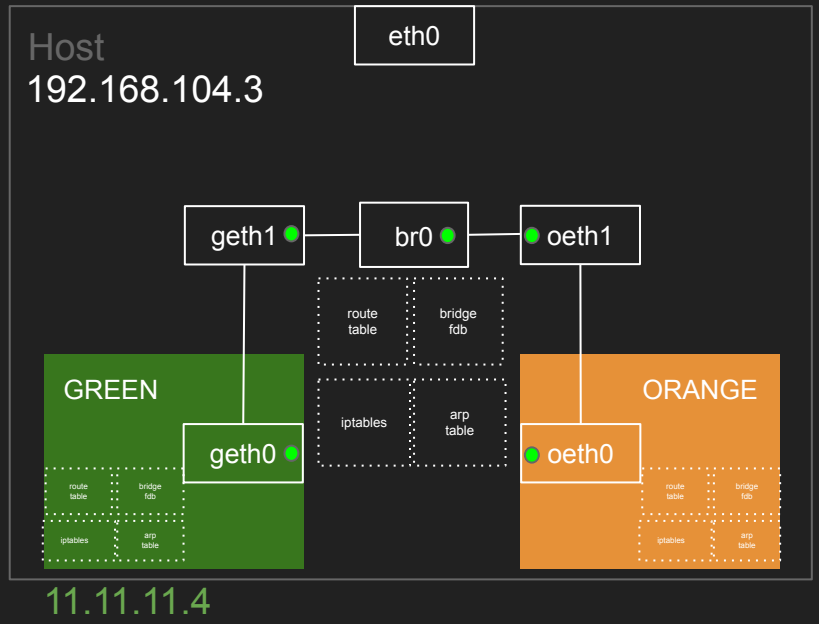


서버 간에도 (가상의) 네트워크 네임스페이스 통신이 가능할까요?



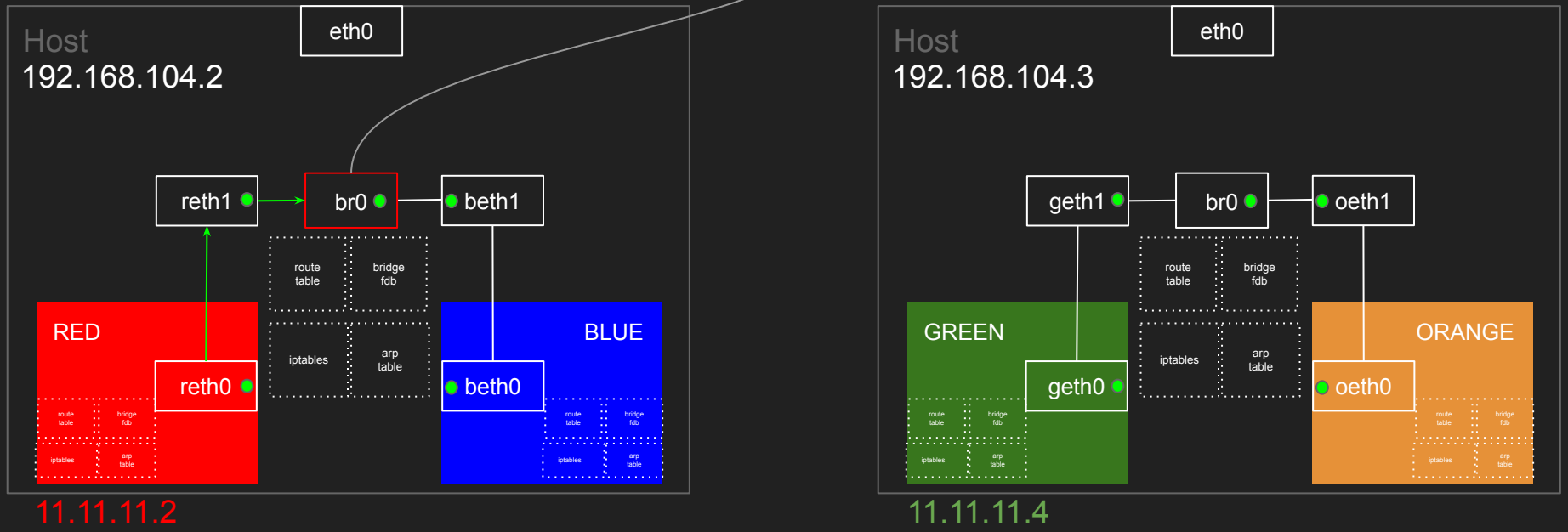
PING 11.11.11.2 → 11.11.11.4





PING 11.11.11.2 → 11.11.11.4

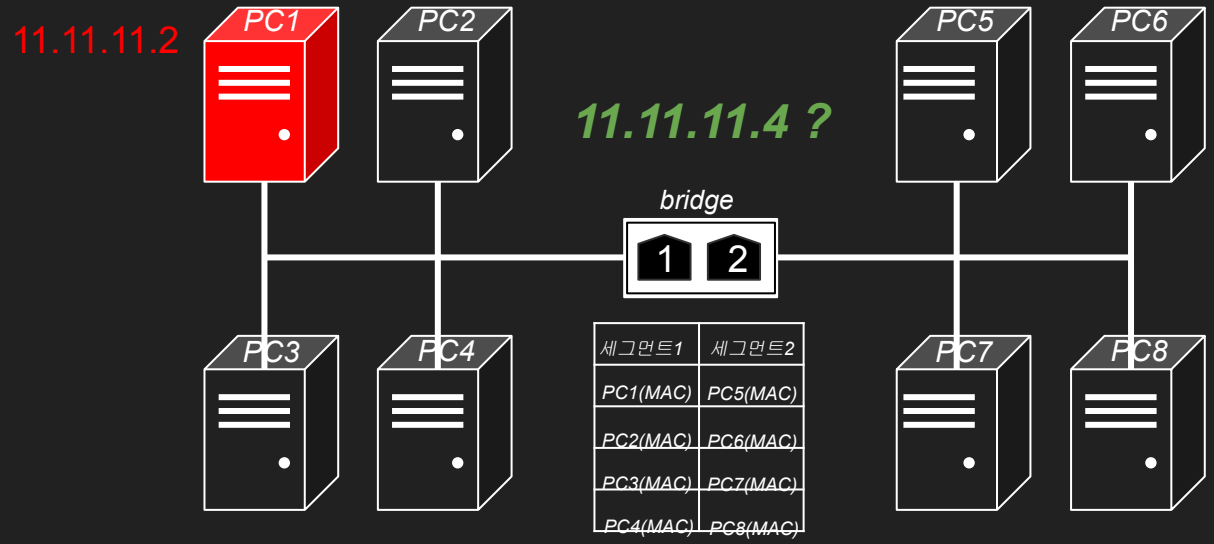
```
listening on br0, link-type EN10MB (Ethernet), capture size 262144 bytes
10:42:32.890245 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
10:42:33.897011 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
10:42:34.920864 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
10:42:35.945195 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
```



Bridge

서버는 포트에 연결되고 포트별로 어떤 세그먼트에 속하는지 관리

```
listening on br0, link-type EN10MB (Ethernet), capture size 262144 bytes
10:42:32.890245 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
10:42:33.897011 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
10:42:34.920864 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
10:42:35.945195 ARP, Request who-has 11.11.11.4 tell 11.11.11.2, length 28
```



ARP broadcasting .. looking for 11.11.11.4

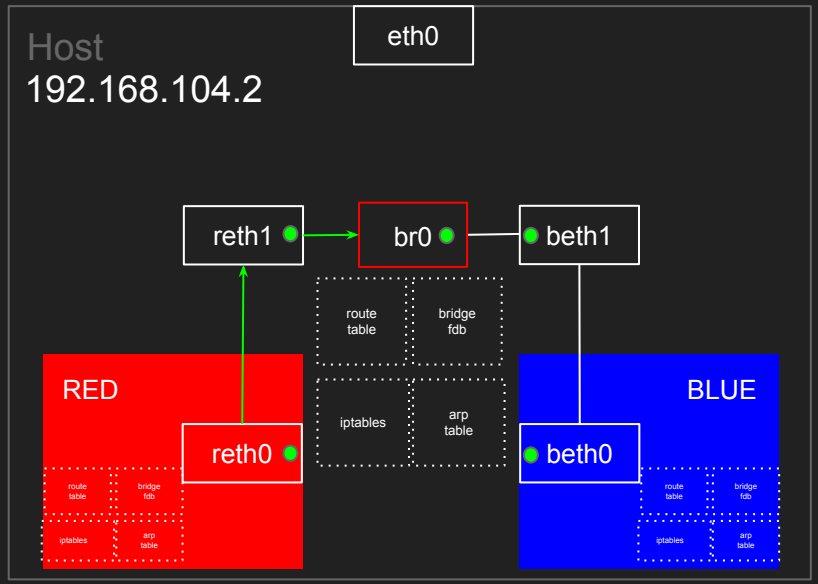
PING 11.11.11.2 → 11.11.11.4

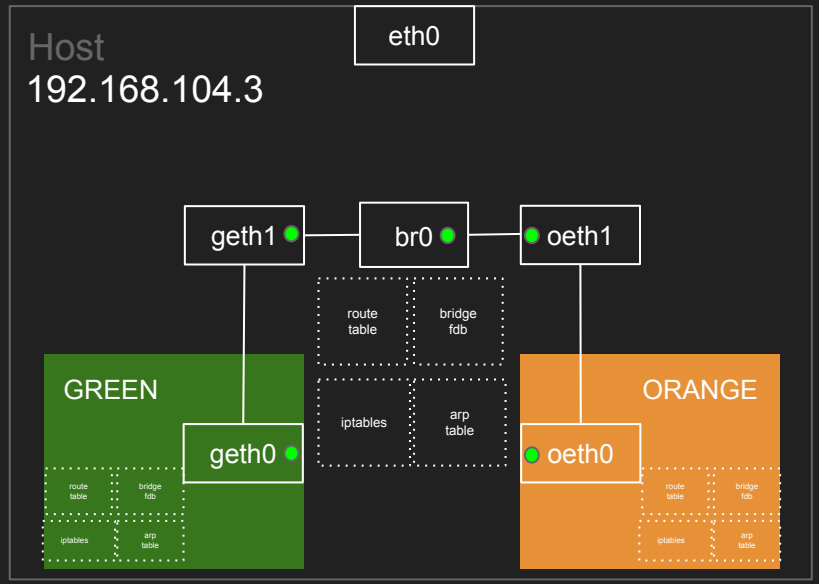
라우트 테이블을 보면 …

ip route

```
default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15 metric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
10.0.2.2 dev eth0 proto dhcp scope link src 10.0.2.15 metric 100
11.11.11.0/24 dev br0 proto kernel scope link src 11.11.11.1
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.104.0/24 dev eth1 proto kernel scope link src 192.168.104.2
```

분명 같은 동네 (11.11.11.0/24) 인데 ?

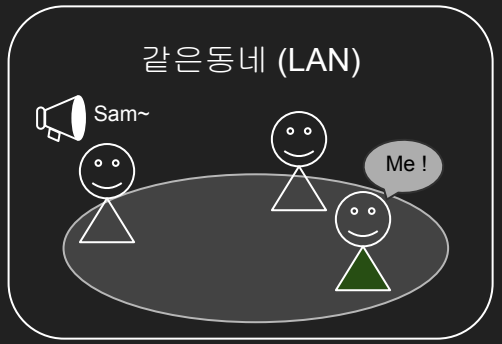


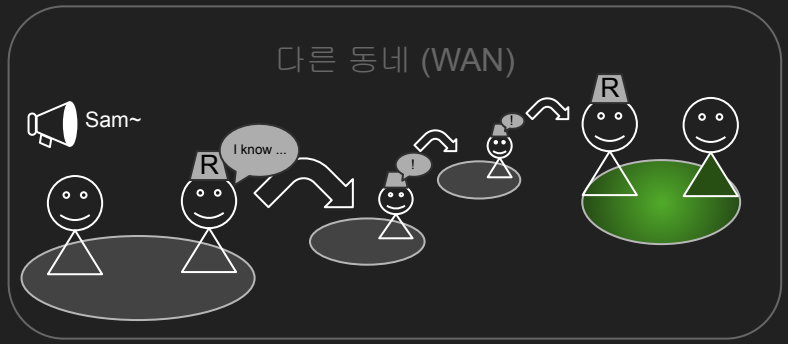


같은 동네 (LAN) ?

연결

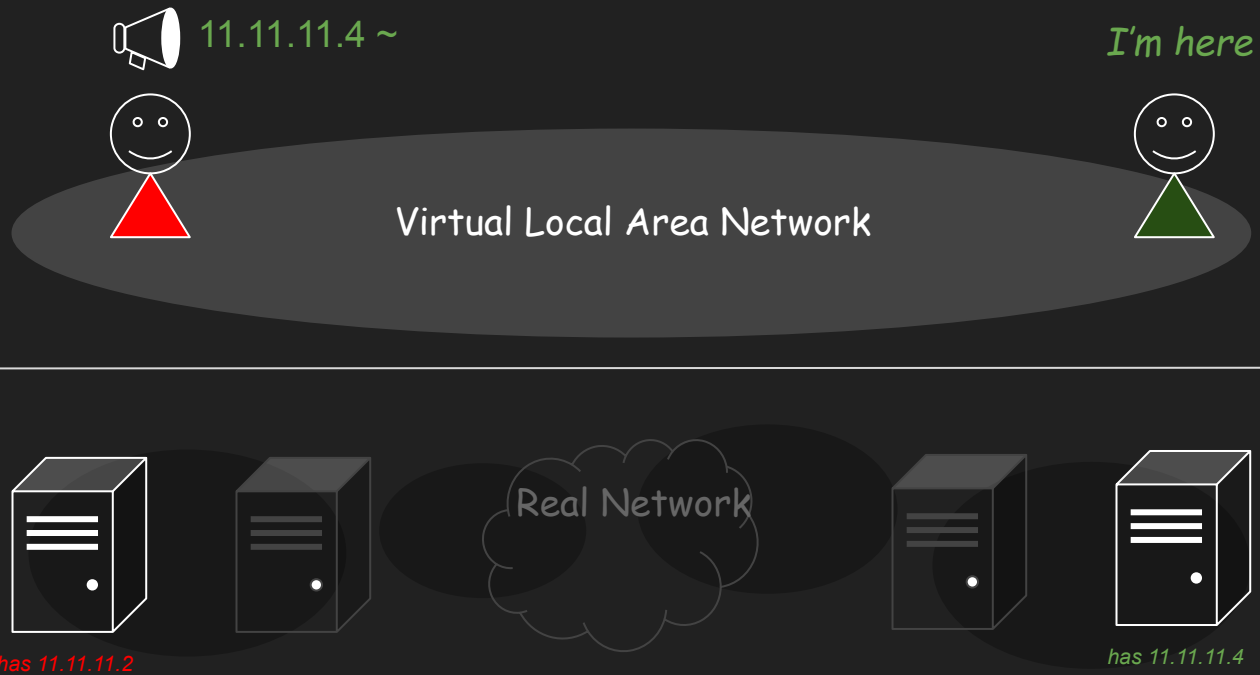
- LAN : Local Area Network (브로드캐스트 도메인)
- WAN : Wide Area Network (라우터로 구분되는 네트워크)



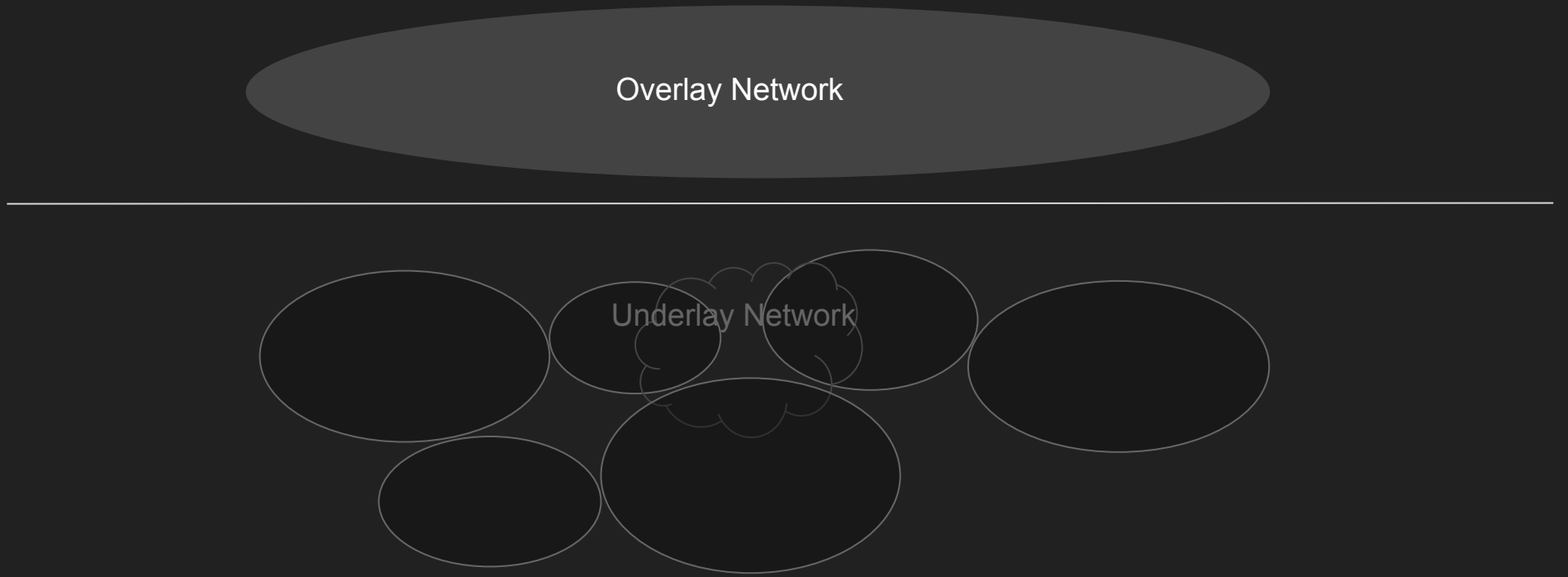


VLAN

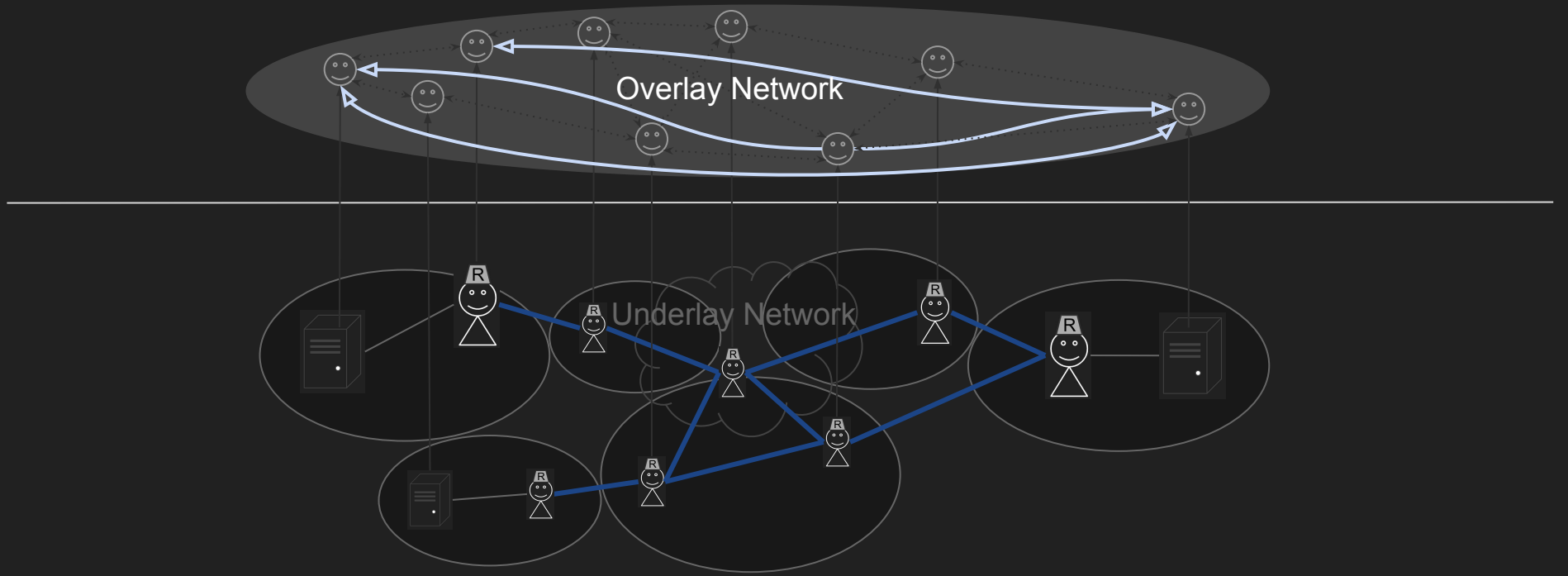
가상의 Local Area Network 가 가능하다면 ?



오버레이 네트워크



오버레이 네트워크 *하부 네트워크 구조가 어찌됐건 "동일한 네트워크 구성을 유지"*



VLAN

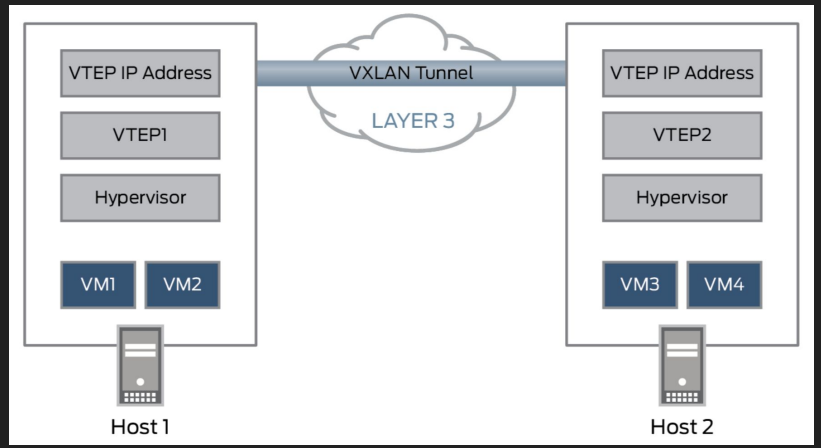
VLAN (Virtual LAN) 오버레이 네트워크 구현 기술 중 하나 …

GRE tunnel, MPLS, VPN , …

VxLAN

Virtual eXtensible LAN

- *VLAN ID (12bit) 4,096개 제약 → ID(24bit) 2^24 개 (16,777,216)*
- *L2 over L3 : UDP 패킷 내부에 L2 프레임을 캡슐화하는 터널링 기술*
- *VTEP (Vxlan Tunnel End Point) : 종단역할. encapsulation / termination*

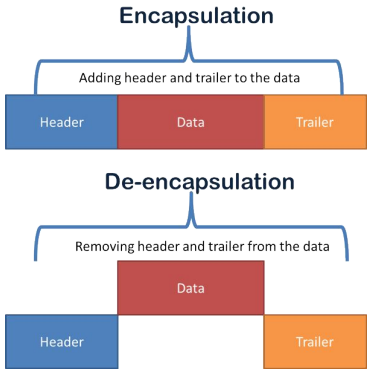


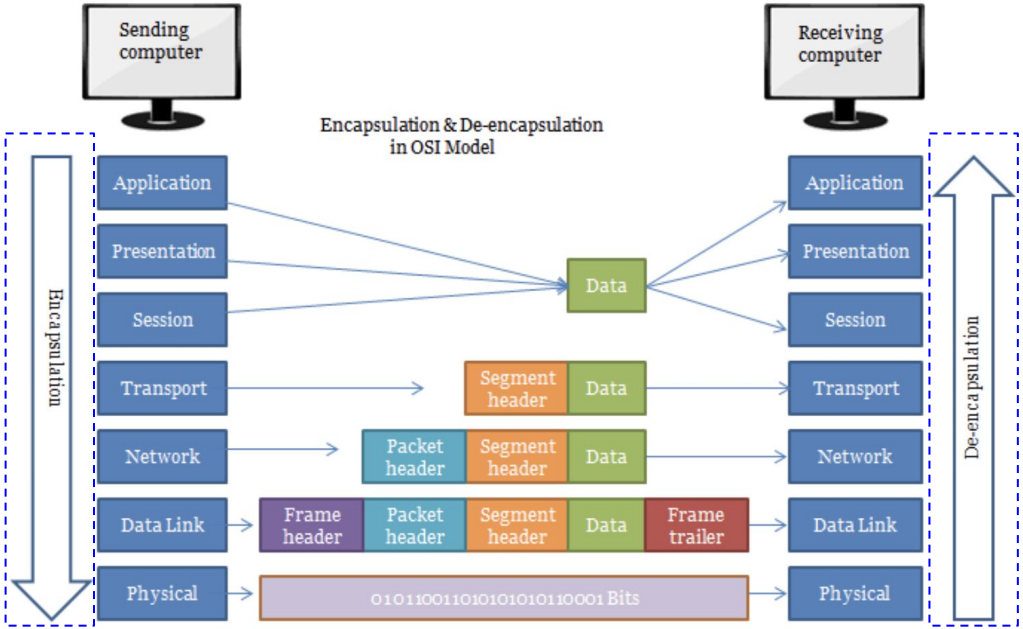
https://atthis.tistory.com/7?category=750237

캡슐화

Network Stack

- 캡슐화 : Layer 구분 / Layer간 통신





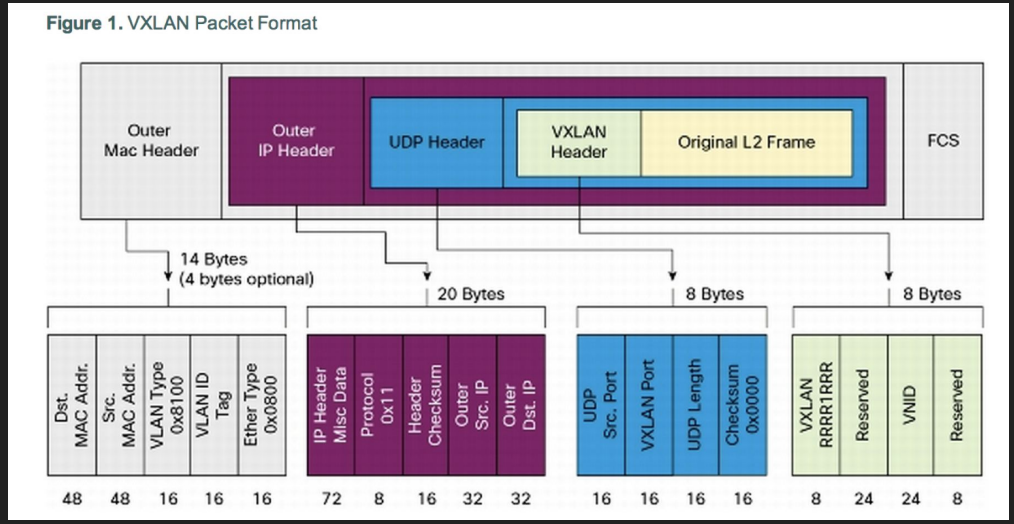
터널링

터널링의 구성요소 :

- 승객 Passenger Protocol : 캡슐화 대상 프로토콜
- 전달 Carrier Protocol : 캡슐화 시킬 프로토콜
- 전송 Transport Protocol : 전달 프로토콜을 끌고 갈 프로토콜

MAC-in-UDP

VXLAN L2 정보 (MAC address)를 L3에 넣어서 통신



VXLAN Packet Size : 50 Bytes (14 + 20 + 8 + 8)

MTU

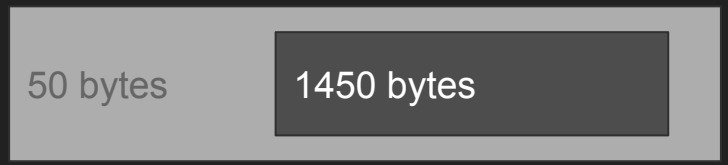
Maximum frame or packet size for a particular network medium. Typically 1500 bytes for Ethernet networks

VXLAN 사용 시 MTU 는 1450 (1500-50) 으로 설정 한다.

** VXLAN Packet Size : 50 Bytes (14 + 20 + 8 + 8)*

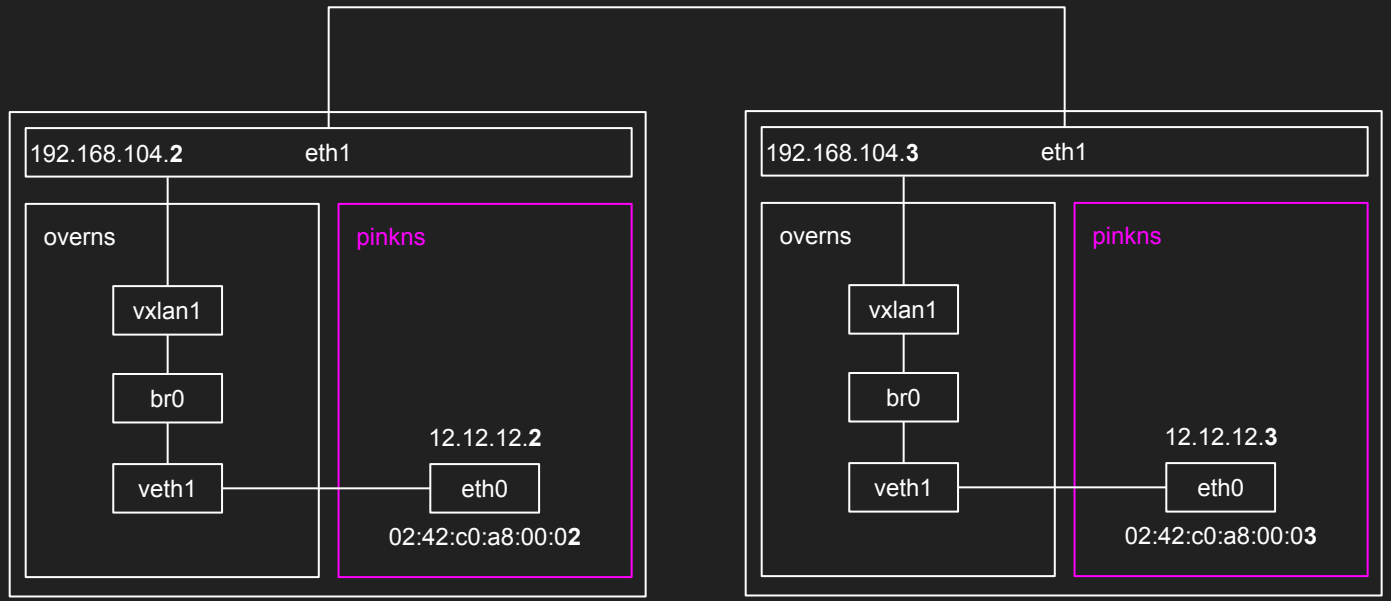
** 왜 50을 빼냐고요? (바로 앞 장의 그림을 다시 한번 봐주세용~)*

*물리 네트워크 통신의 **decapsulation** 과정에서 **50 bytes** 가 사용되었고*
*VXLAN이 가상 네트워크 통신을 위한 나머지 **1450 bytes**를 전달하게 됩니다.*



(실습1) 오버레이
네트워크

오늘 그릴 그림



(실습1) 오버레이 네트워크

네트워크 네임스페이스 생성

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip netns add overns
# ip netns add pinkns
```

(실습1) 오버레이 네트워크

네트워크 네임스페이스로 veth pair 이동

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip link add dev veth1 mtu 1450 netns overns type veth peer name eth0 mtu 1450 netns pinkns
```

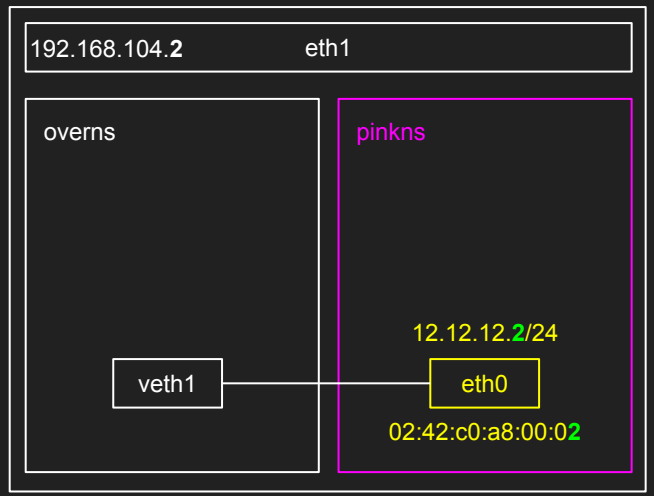
(실습1) 오버레이 네트워크

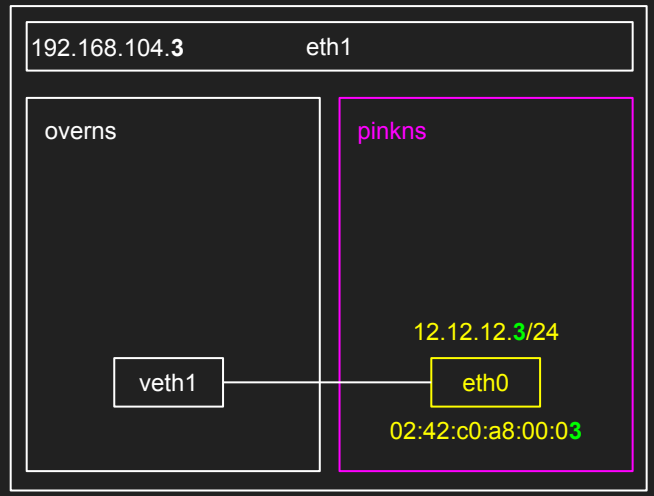
veth pair 에 MAC / IP address 설정

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip netns exec pinkns ip link set dev eth0 address <MAC ADDRESS - 02:42:c0:a8:00:0?>
# ip netns exec pinkns ip addr add dev eth0 <IP/CIDR - 12.12.12.?/24>
```

? : 각 창에 알맞은 숫자로 넣어주세요



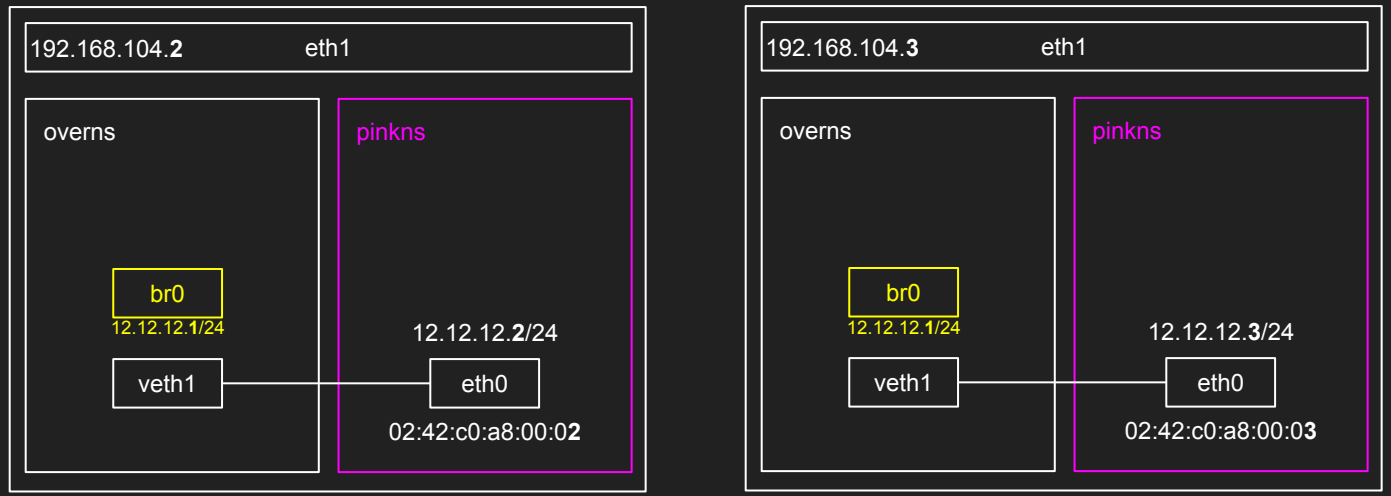


(실습1) 오버레이 네트워크

bridge (br0) 추가 및 설정

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip netns exec overns ip link add dev br0 type bridge
# ip netns exec overns ip addr add dev br0 12.12.12.1/24
```

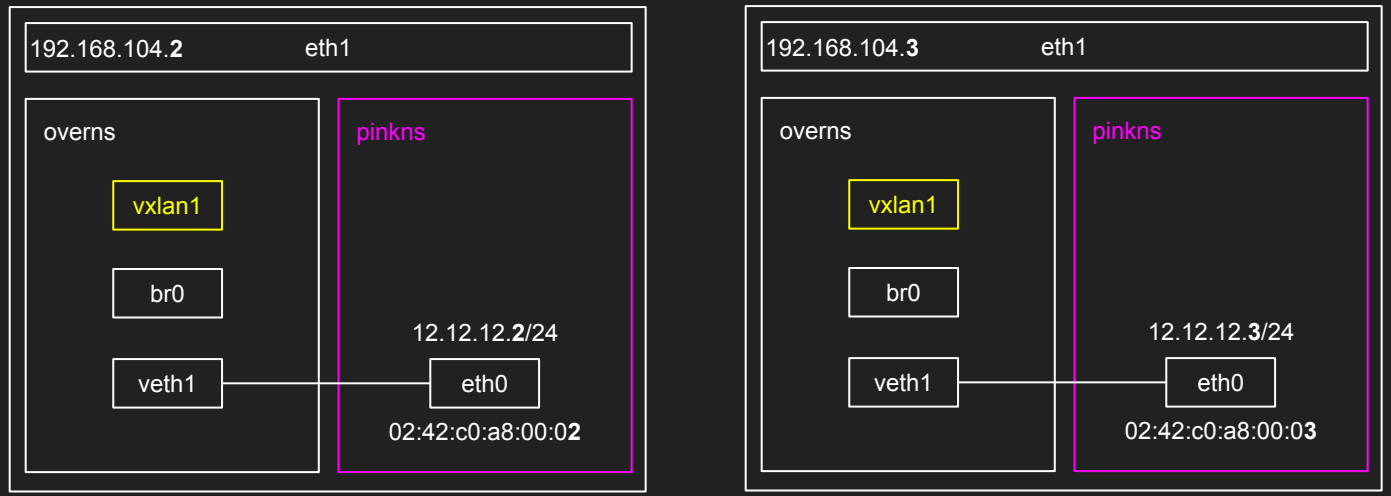


(실습1) 오버레이 네트워크

vxlan 생성 및 설정

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip link add dev vxlan1 netns overns type vxlan id 42 proxy learning dstport 4789
```



(실습1) 오버레이 네트워크

vxlan 생성 및 설정

ip link add dev vxlan1 netns overns type vxlan id 42 proxy learning dstport 4789

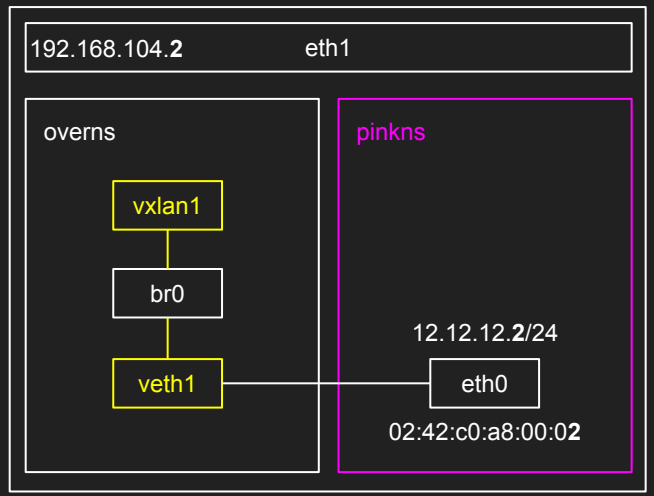
id 42 ~ VNI . vxlan endpoint 식별
proxy ~ vxlan 이 arp 쿼리에 응답하도록 허용
learning ~ bridge fdb entry 자동 갱신 허용
dstport 4789 ~ UDP port for 터널링

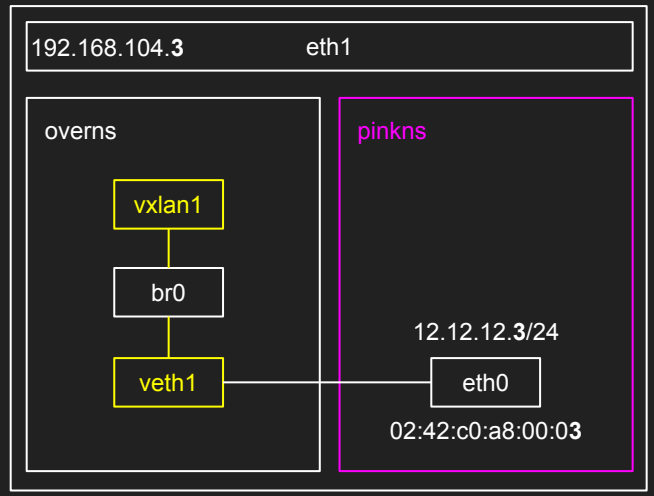
(실습1) 오버레이 네트워크

veth1, vxlan1을 br0에 연결

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip netns exec overns ip link set veth1 master br0
# ip netns exec overns ip link set vxlan1 master br0
```



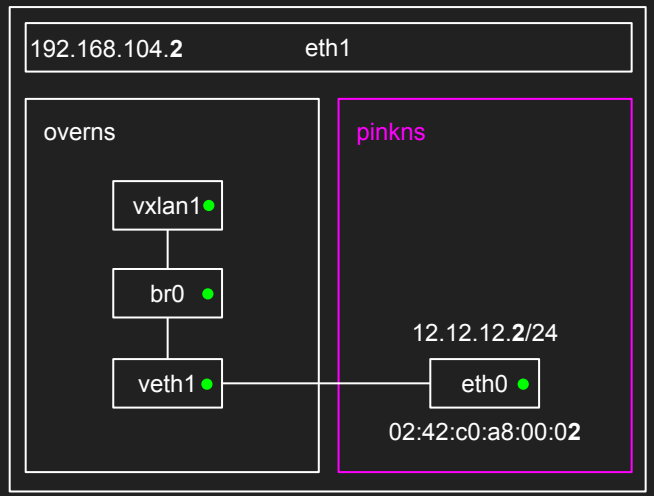


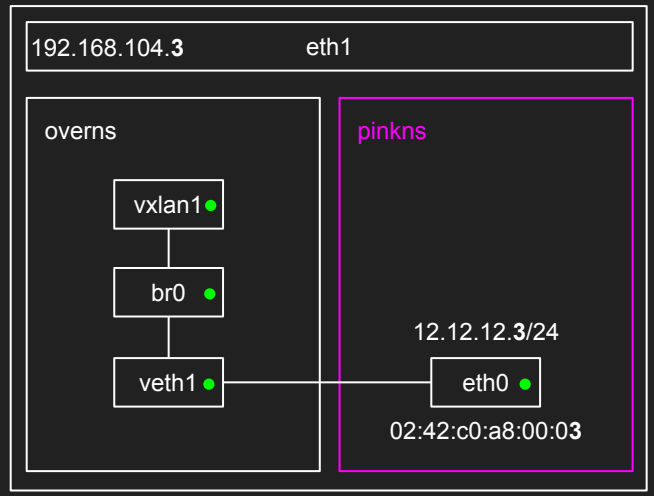
(실습1) 오버레이 네트워크

각 virtual interface 의 전원 On ~

터미널 #1 (192.168.104.2) + 터미널 #2 (192.168.104.3)

```
# ip netns exec overns ip link set br0 up
# ip netns exec overns ip link set vxlan1 up
# ip netns exec overns ip link set veth1 up
# ip netns exec pinkns ip link set eth0 up
```





(실습1) 오버레이 네트워크

생성한 네트워크 네임스페이스에 들어가 봅시다

터미널 #1 (192.168.104.2)

```
# nsenter --net=/var/run/netns/pinkns
```

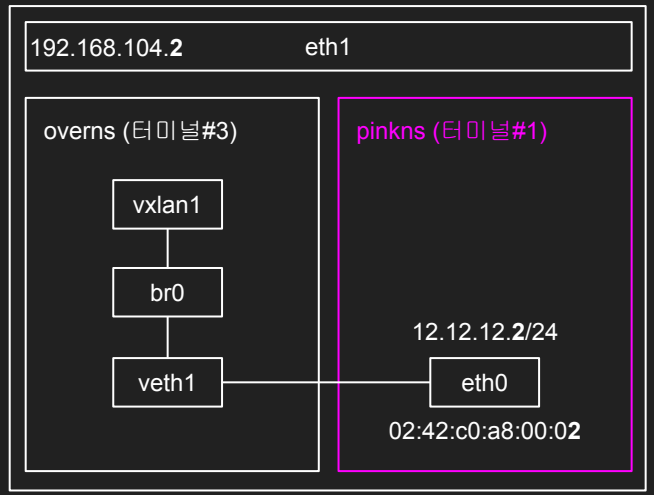
터미널 #2 (192.168.104.3)

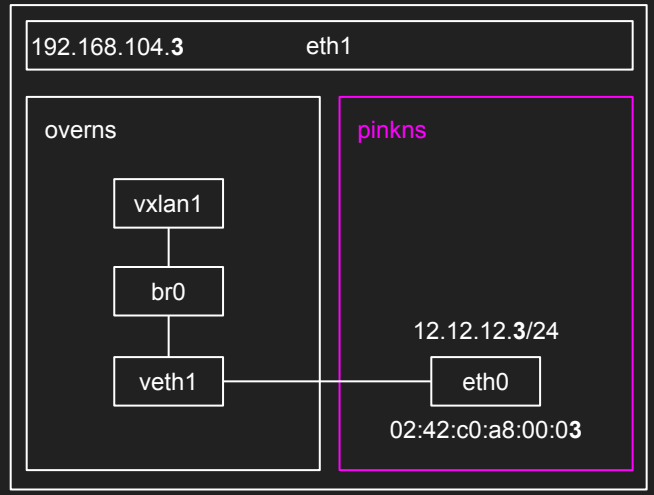
```
# tcpdump -i eth1
```

하나 더

터미널 #3 (192.168.104.2)

```
# nsenter --net=/var/run/netns/overns
```





(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

터미널 #1 (pinkns@192.168.104.2)

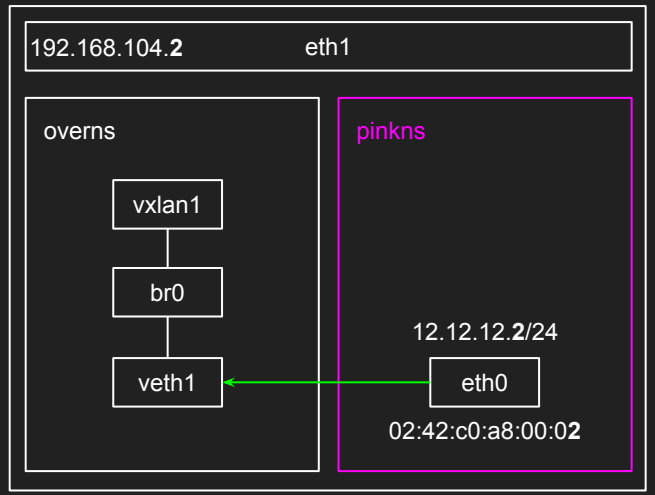
```
# ping 12.12.12.3
```

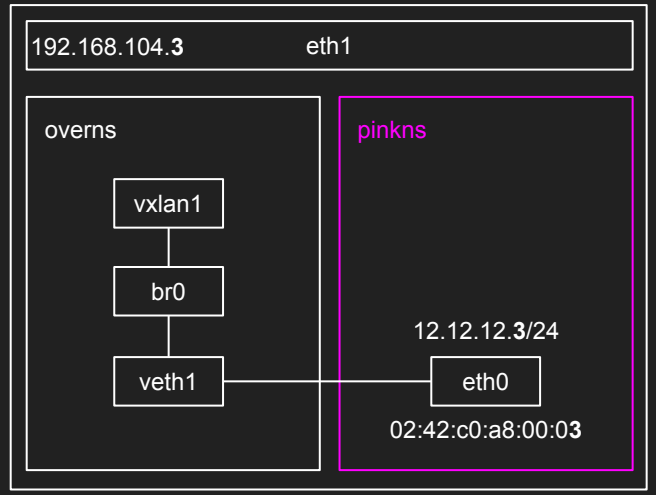
터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i br0
```





(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요?

터미널 #1 (pinkns@192.168.104.2)

```
# ping 12.12.12.3
```

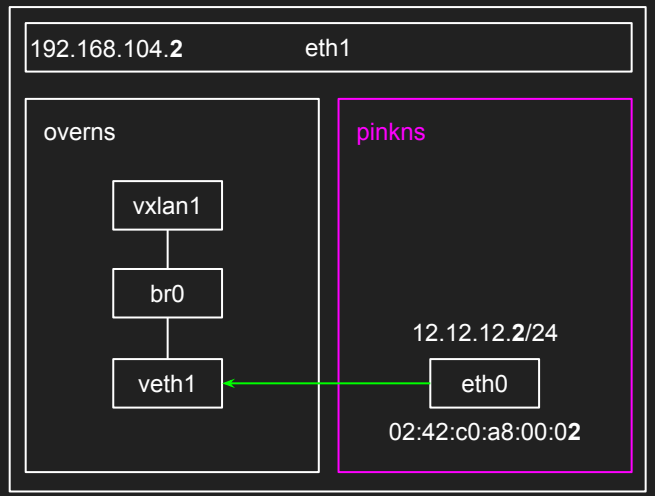
터미널 #2 (192.168.104.3)

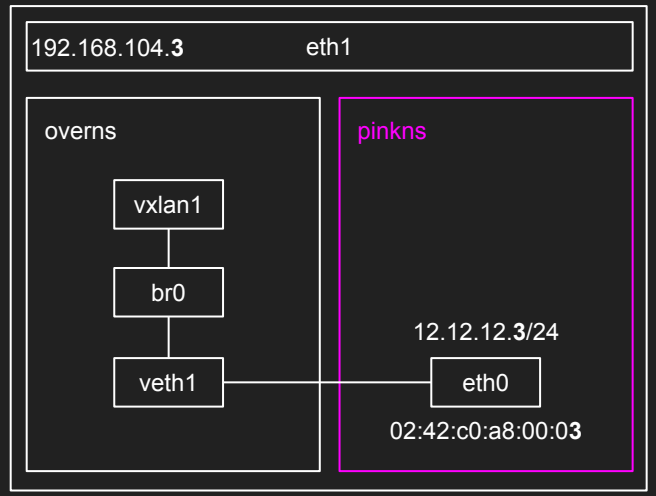
```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i br0
ARP, Request who-has 12.12.12.3 tell 12.12.12.2, length 28
```

?

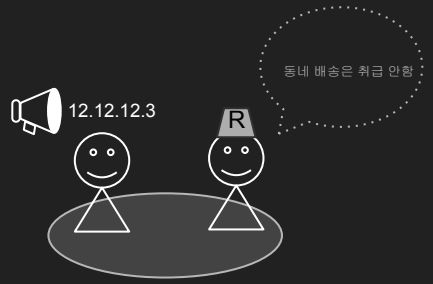


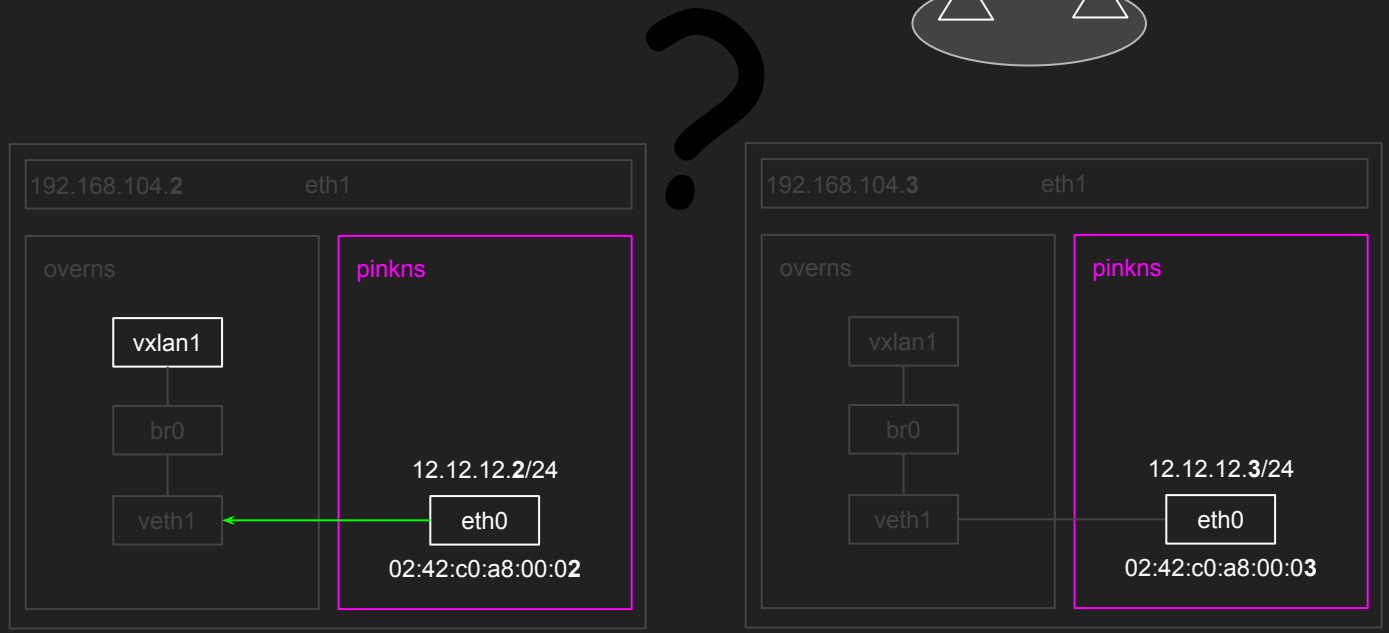


(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요?

12.12.12.3은 응답할 수가 없습니다.

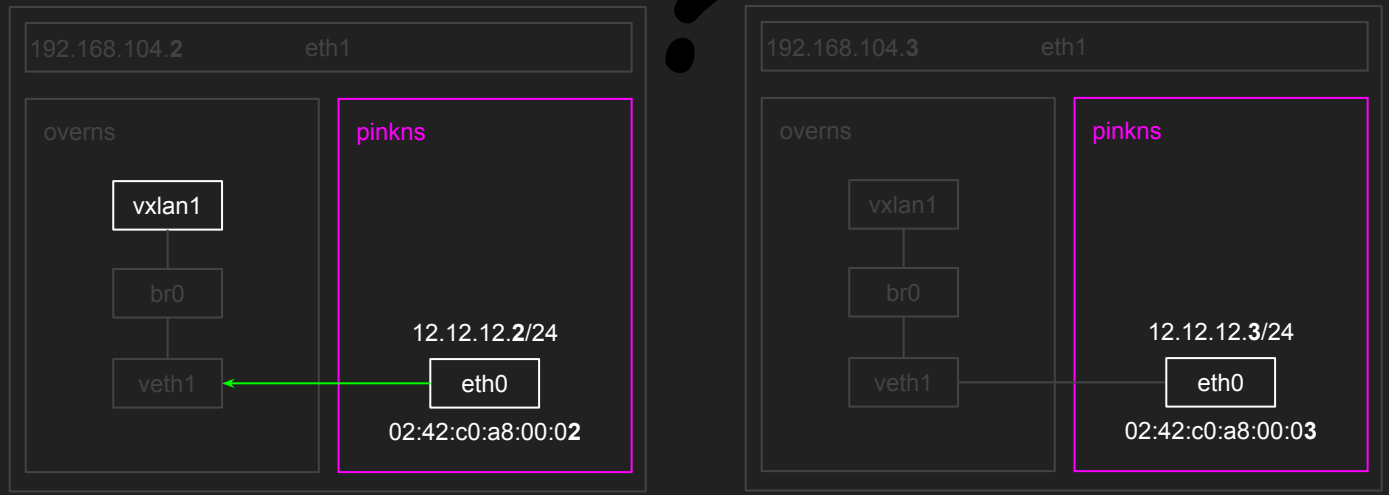




(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요?

12.12.12.3은 누가 전달해 줄 수 있을까요?



(실습1) 오버레이 네트워크

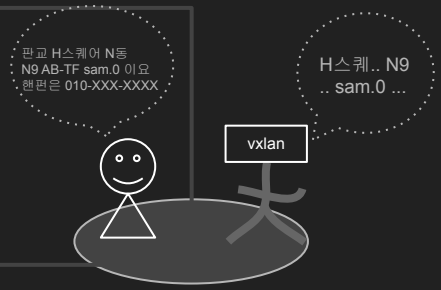
12.12.12.3의 MAC주소가 뭐에요?

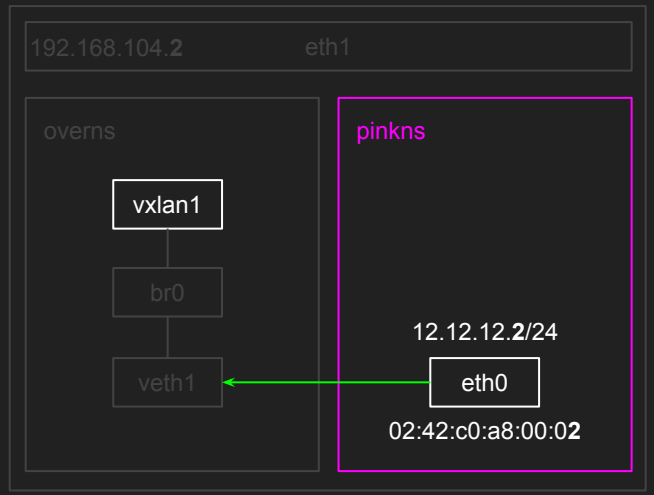
터미널 #3 (overns@192.168.104.2)

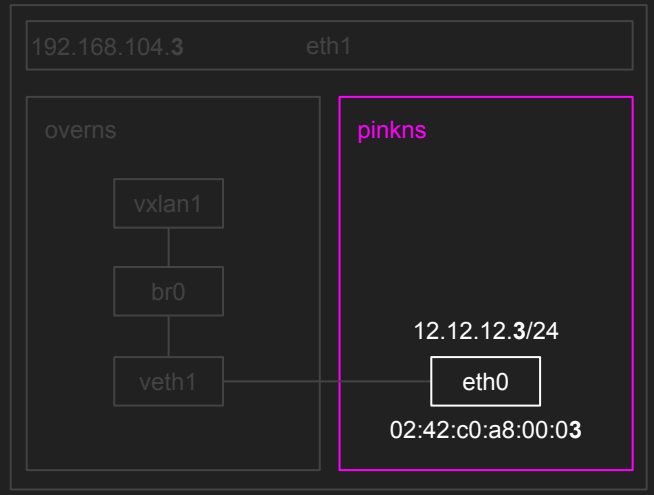
```
# ip neigh show

# ip neigh add 12.12.12.3 lladdr 02:42:c0:a8:00:03 dev vxlan1

# ip neigh show
12.12.12.3 dev vxlan1 lladdr 02:42:c0:a8:00:03 PERMANENT
```

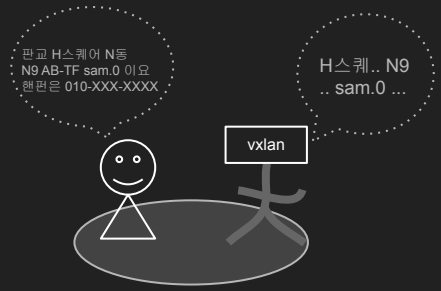






(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요?

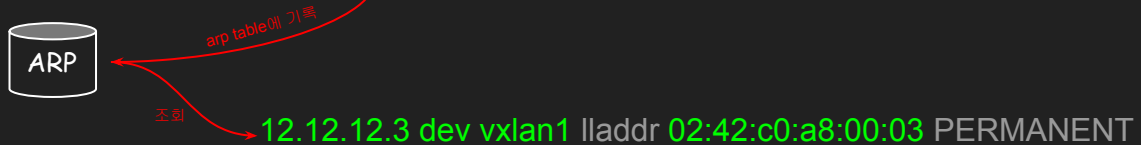


목적지 IP 주소

목적지 MAC 주소

전송을 담당할 디바이스

```
# ip neigh add 12.12.12.3 lladdr 02:42:c0:a8:00:03 dev vxlan1
```



lladdr? L2 주소

영구 사용

STATE NAME	VALUE	BRIEF MEANING
NONE	00000000	Pseudo state used while an ARP entry is initially created or just before it is removed
INCOMPLETE	00000001	First ARP request sent
REACHABLE	00000002	ARP response is received
STALE	00000004	ARP response is not received within expected time
DELAY	00000008	Schedule ARP request
PROBE	00000010	Actively sending ARP requests to try and resolve the address
FAILED	00000020	Not managed to resolve ARP within the maximum configured number of probes
NOARP	00000040	Device does not support ARP e.g. IPsec interface
PERMANENT	00000080	Statically configured ARP entry

arp 상태 참고 : https://ebt-forti.tistory.com/m/145?category=849281

(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요?

터미널 #1 (pinkns@192.168.104.2)

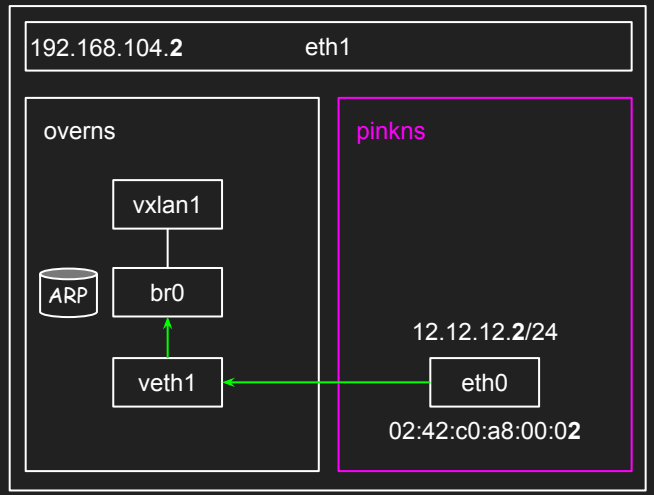
```
# ping 12.12.12.3
```

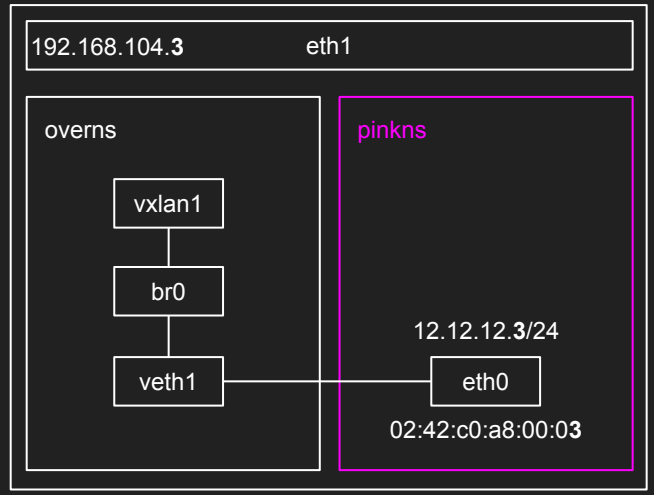
터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i br0
. . .
ARP, Reply 12.12.12.3 is-at 02:42:c0:a8:00:03 (oui Unknown), length 28
```





(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요?

터미널 #1 (pinkns@192.168.104.2)

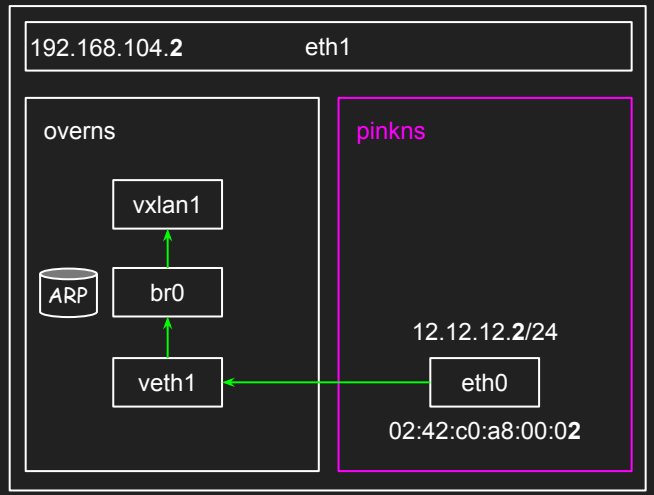
```
# ping 12.12.12.3
```

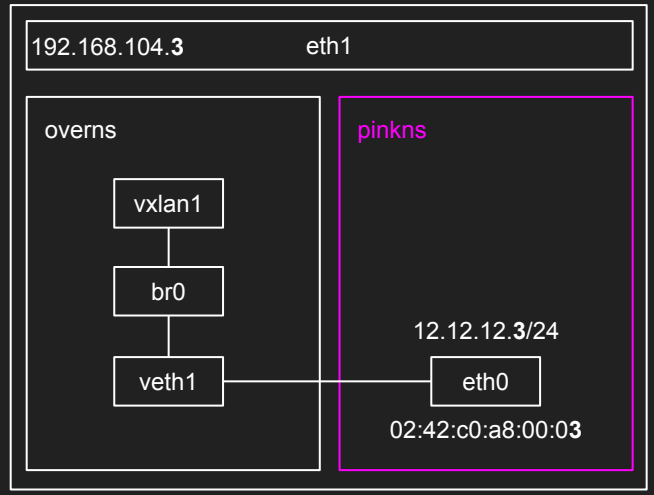
터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
. . .
ARP, Reply 12.12.12.3 is-at 02:42:c0:a8:00:03 (oui Unknown), length 28
```





(실습1) 오버레이 네트워크

12.12.12.3의 MAC주소가 뭐에요? 02:42:c0:a8:00:03

터미널 #1 (pinkns@192.168.104.2)

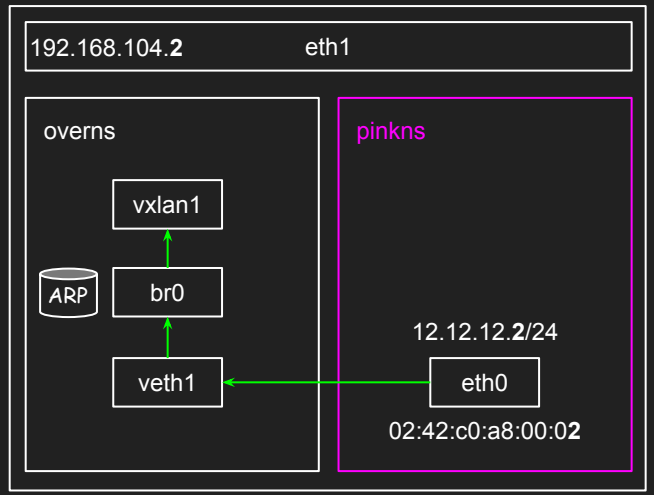
```
# ip neigh show
12.12.12.3 dev eth0 lladdr 02:42:c0:a8:00:03 REACHABLE
```

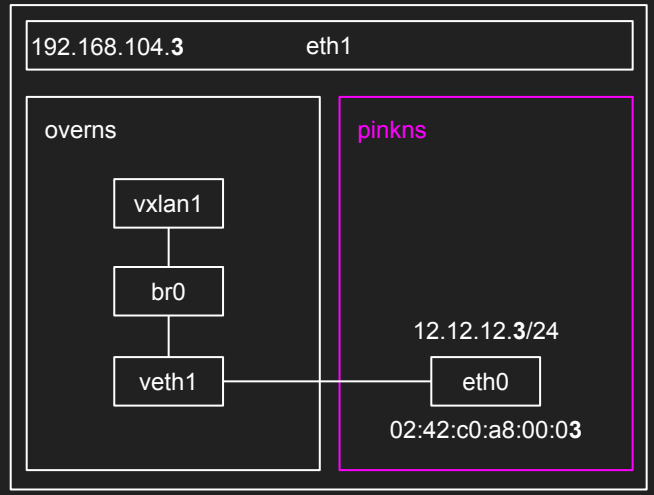
터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
. . .
ARP, Reply 12.12.12.3 is-at 02:42:c0:a8:00:03 (oui Unknown), length 28
```





(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

No ICMP reply ?

터미널 #1 (pinkns@192.168.104.2)

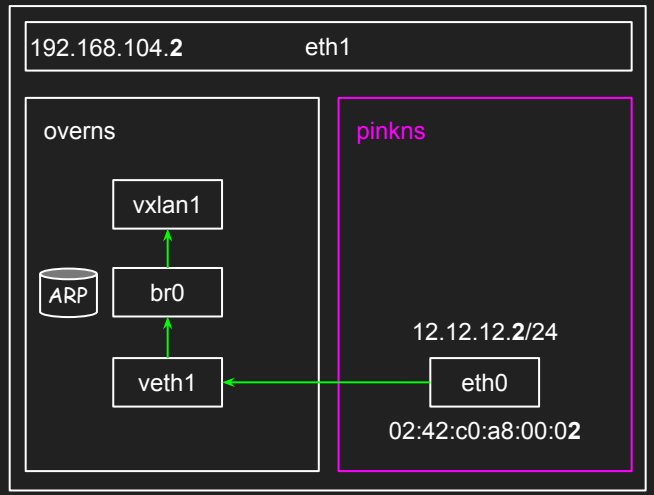
```
# ping 12.12.12.3
```

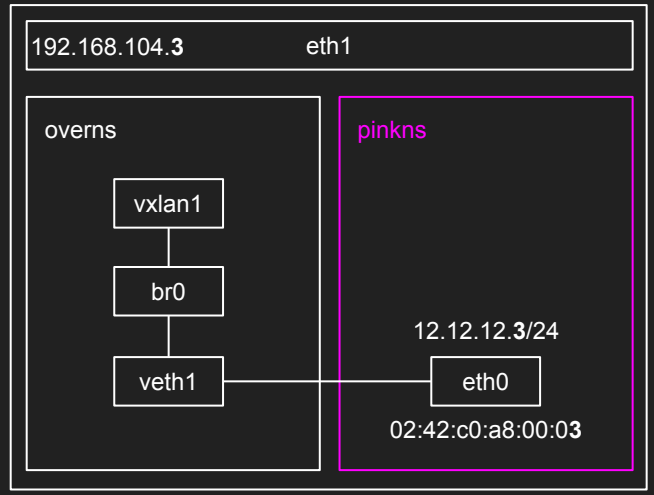
터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
. . .
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2648, seq 60, length 64
```





(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

No ICMP reply ?

ICMP (Internet Control Message Protocol)

IP 동작 진단/제어에 사용되고 오류에 대한 응답을 source IP에 제공

- L3 (Network Layer) 프로토콜
- 인터넷/통신 상황 report , 오류 보고, 위험상황에 대한 경보 등에 사용
- ping : destination host 작동여부 / 응답 시간 측정
- tracert : destination routing 경로 추적

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

No ICMP reply ?

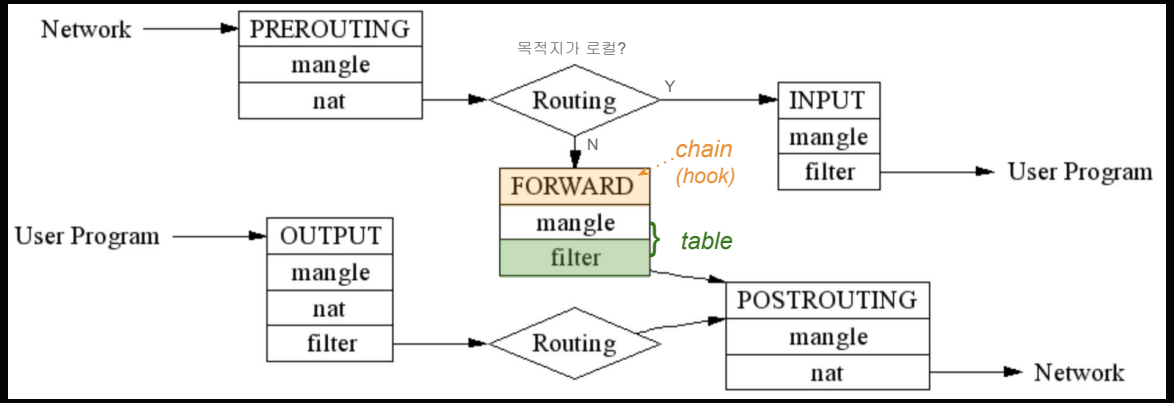
iptables

FOWARD : NF_IP_FORWARD (hook)에 등록된 체인

ㄴ NF_IP_FORWARD : incoming 패킷이 다른 호스트로 포워딩되는 경우에 트리거되는 netfilter hook

filter (table) : 패킷을 목적지로 전송 여부를 결정

Packet flow



(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

No ICMP reply ?

iptables forward 룰을 확인해 봅니다

터미널 #3 (overns@192.168.104.2)

```
# iptables -t filter -L | grep policy

Chain INPUT (policy ACCEPT)
Chain FORWARD (policy ACCEPT)
Chain OUTPUT (policy ACCEPT)
```

Q) 왜 호스트가 아닌 overns의 iptables을 확인하였을까요?

네~ br0가 호스트가 아닌 overns 에 있기 때문입니다.

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

No Reply ICMP ?

터미널 #1 (pinkns@192.168.104.2)

```
# ping 12.12.12.3
```

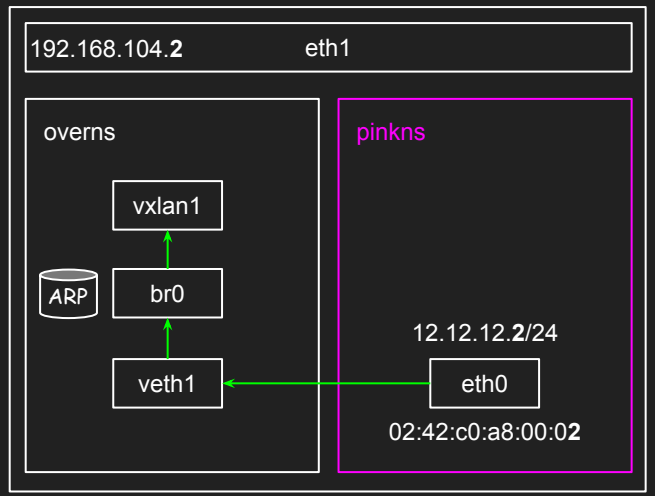
터미널 #2 (192.168.104.3)

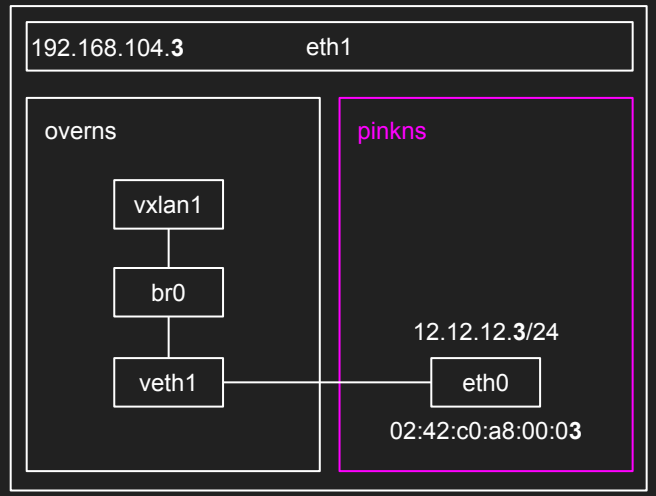
```
# tcpdump -i eth1
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
. . .
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2648, seq 60, length 64
```

?





(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

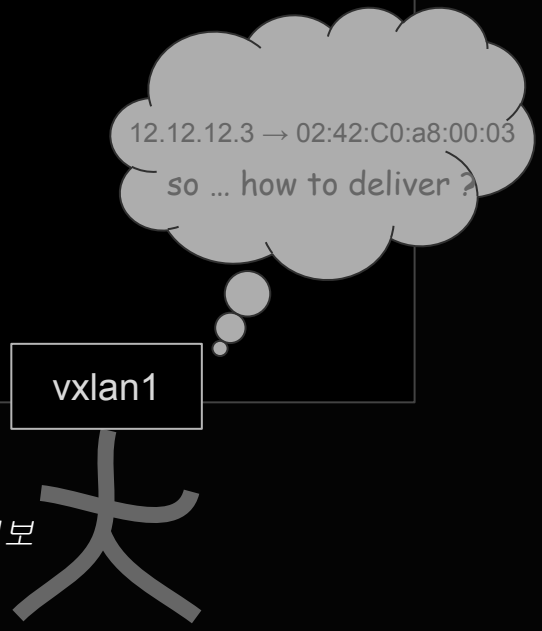
No Reply ICMP ?

터미널 #3 (overns@192.168.104.2)

```
# ip route
12.12.12.0/24 dev br0 proto kernel scope link src 12.12.12.1

# ip neigh show
12.12.12.3 dev vxlan1 lladdr 02:42:c0:a8:00:03 PERMANENT

# bridge fdb show
. . .
02:42:c0:a8:00:03 dev vxlan1 master br0
. . .
```



route table ~ 12.12.12.0 대역은 br0가 gateway

arp table ~ 12.12.12.3 → 02:42:c0:a8:00:03

bridge fdb ~ 02:42:c0:a8:00:03 → ***vxlan1****가 처리하고* ***br0****에 연결돼 있다는 정보*

뭔가 정보가 부족한가 봅니다

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

No Reply ICMP ?

vxlan 에게 정확한 길을 상세히 알려 줍시다

터미널 #3 (overns@192.168.104.2)

VTEP ID
패킷을 수령할
엔드포인트

```
# bridge fdb add 02:42:c0:a8:00:03 dev vxlan1 self dst 192.168.104.3 vni 42 port 4789
```

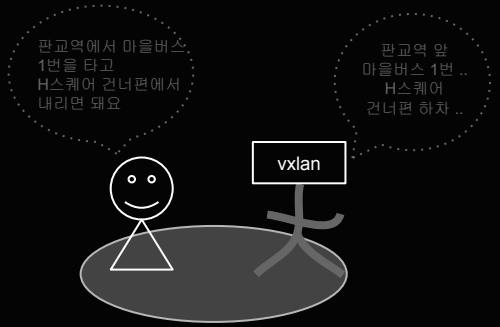
실제 목적지의
물리 IP 주소

실제 목적지의
물리 port

FDB

```
# bridge fdb show | grep 02:42:c0:a8:00:03
02:42:c0:a8:00:03 dev vxlan1 dst 192.168.104.3 link-netnsid 0 self permanent
```

"02:42:c0:a8:00:03은 가상의 MAC address로
vxlan이 잘 포장해서 퀵으로 보내야 하므로
MAC주소를 처리할 수신처(호스트) IP를 적어주어야 합니다."



(실습1) 오버레이 네트워크

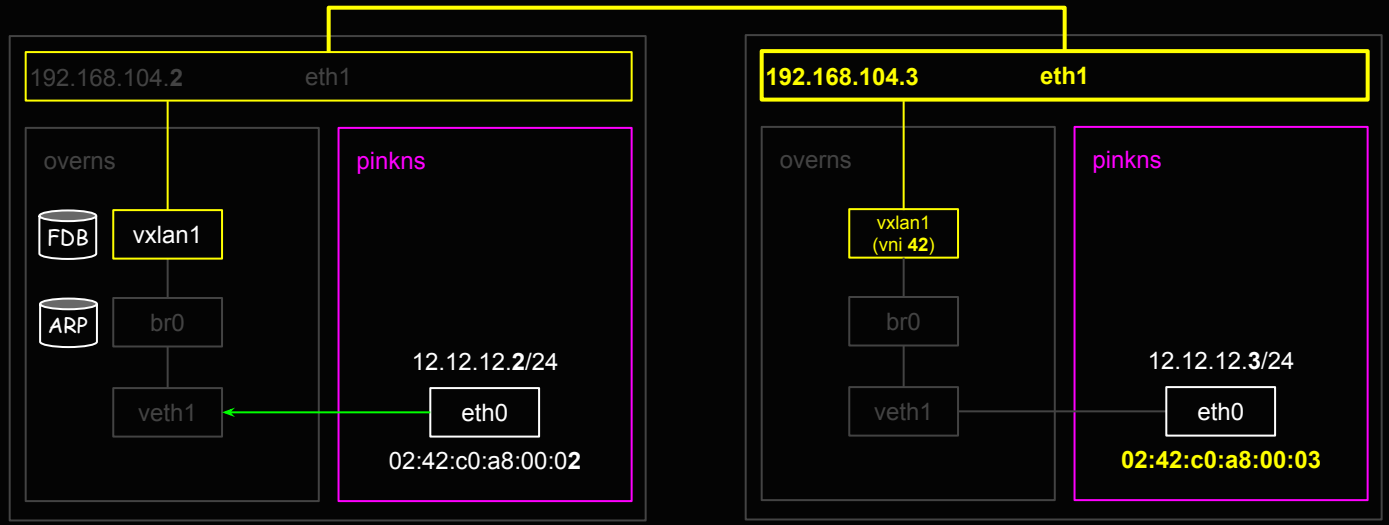
Ping 12.12.12.2 → 12.12.12.3

No Reply ICMP ?

*vxlan 에게 정확한 길을 **상세히** 알려 줍시다* (FDB)

터미널 #3 (overns@192.168.104.2)

```
# bridge fdb add 02:42:c0:a8:00:03 dev vxlan1 self dst 192.168.104.3 vni 42 port 4789
. . .
02:42:c0:a8:00:03 dev vxlan1 dst 192.168.104.3 link-netnsid 0 self permanent
```



(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

오.. 터미널#2에 패킷이 보입니다

터미널 #1 (pinkns@192.168.104.2)

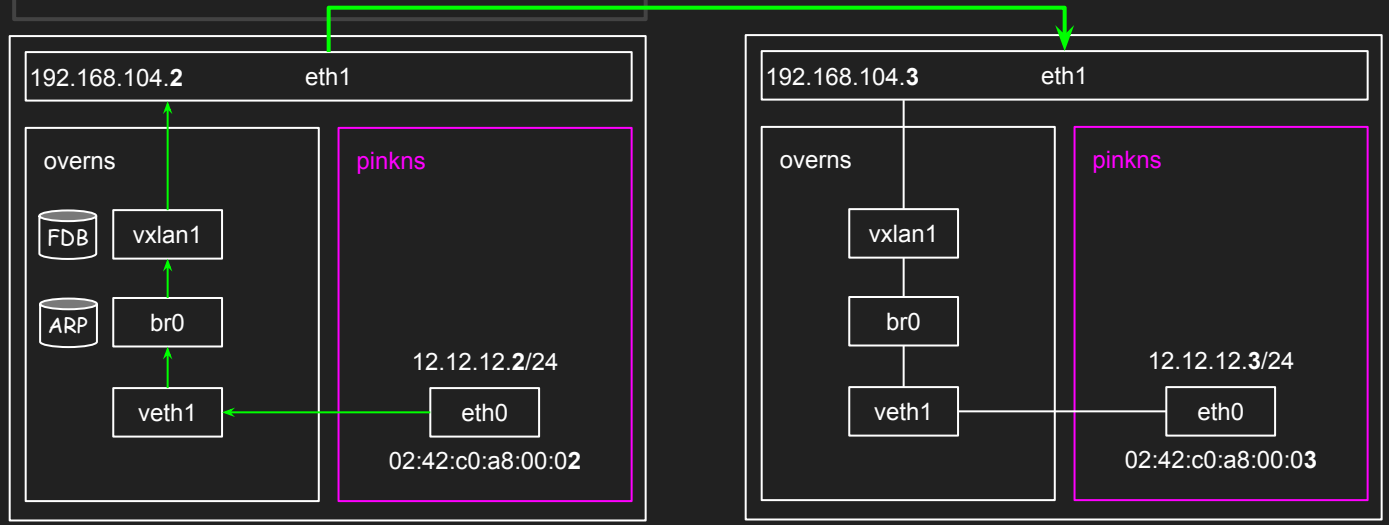
```
# ping 12.12.12.3
```

터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
...
IP 192.168.104.2.39783 > ubuntu1804-2.4789: VXLAN, flags [I] (0x08), vni 42
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2815, seq 3082, length 64
...
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
...
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2648, seq 60, length 64
```



(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

But … No ICMP reply ...

터미널 #1 (pinkns@192.168.104.2)

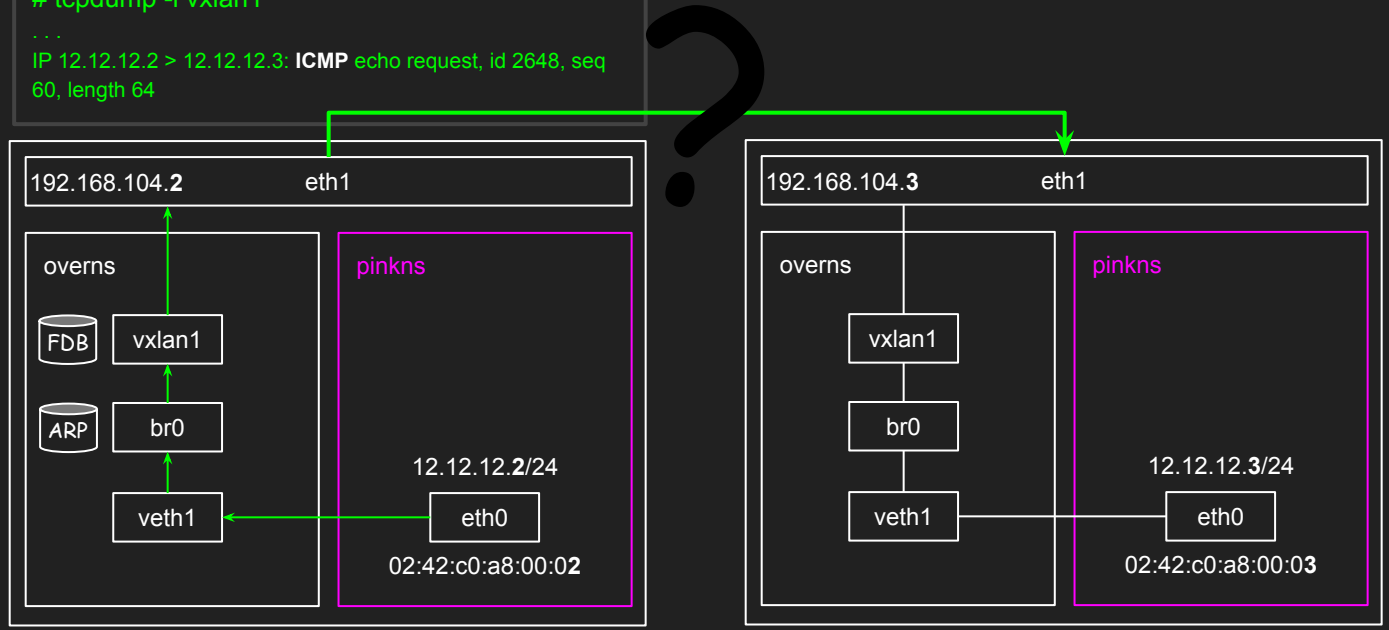
```
# ping 12.12.12.3
```

터미널 #2 (192.168.104.3)

```
# tcpdump -i eth1
...
IP 192.168.104.2.39783 > ubuntu1804-2.4789: VXLAN, flags [I] (0x08), vni 42
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2815, seq 3082, length 64
...
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
...
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2648, seq 60, length 64
```



(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

자 .. 이제 192.168.104.3 을 들여다 볼 차례입니다

터미널 #2의 overns 로 nsenter 합니다.

터미널 #2 (192.168.104.3)

```
# nsenter --net=/var/run/nents/overns
```

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

"감" 오시나요 ?

터미널 #1 (pinkns@192.168.104.2)

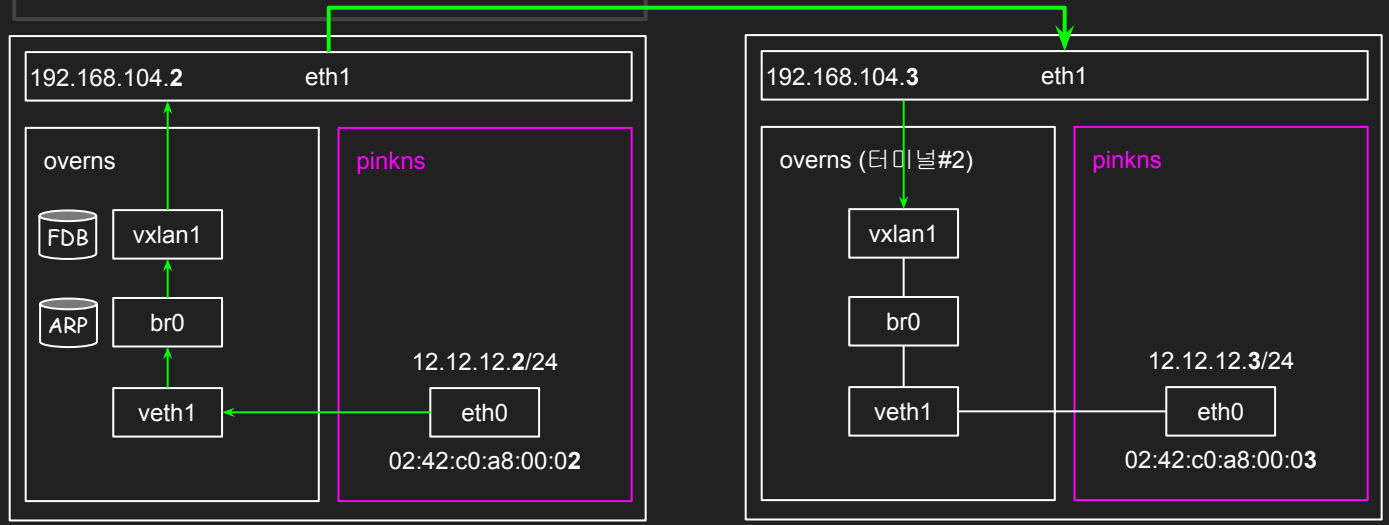
```
# ping 12.12.12.3
```

터미널 #2 (overns@192.168.104.3)

```
# tcpdump -i vxlan1
…
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2833, seq 288, length 64
ARP, Request who-has 12.12.12.2 tell 12.12.12.3, length 28
…
```

터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
. . .
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2648, seq 60, length 64
```



(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

"감" 오시나요 ?

터미널 #2 (overns@192.168.104.3)

```
# tcpdump -i vxlan1
…
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2833, seq 288, length 64
ARP, Request who-has 12.12.12.2 tell 12.12.12.3, length 28
…
```

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

"감" 오시나요 ?

터미널 #2 (overns@192.168.104.3)

```
# ip neigh show

# bridge fdb
02:42:c0:a8:00:02 dev vxlan1 dst 192.168.104.2 link-netnsid 0 self

# ip route
12.12.12.0/24 dev br0 proto kernel scope link src 12.12.12.1
```

ARP 없고

FDB는 있고

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

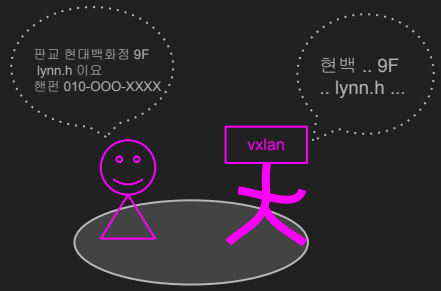
“감” 오시나요 ?

터미널 #2 (overns@192.168.104.3)

```
# ip neigh add 12.12.12.2 lladdr 02:42:c0:a8:00:02 dev vxlan1
```

(실습1) 오버레이 네트워크

12.12.12.2의 MAC주소가 뭐에요?



목적지 IP 주소

목적지 MAC 주소

전송을 담당할 디바이스

```
# ip neigh add 12.12.12.2 lladdr 02:42:c0:a8:00:02 dev vxlan1
```

ARP

arp table에 기록

조회

```
12.12.12.2 dev vxlan1 lladdr 02:42:c0:a8:00:02 PERMANENT
```

lladdr? L2 주소

영구 사용

(실습1) 오버레이 네트워크

Ping 12.12.12.2 → 12.12.12.3

성공 ~

터미널 #1 (pinkns@192.168.104.2)

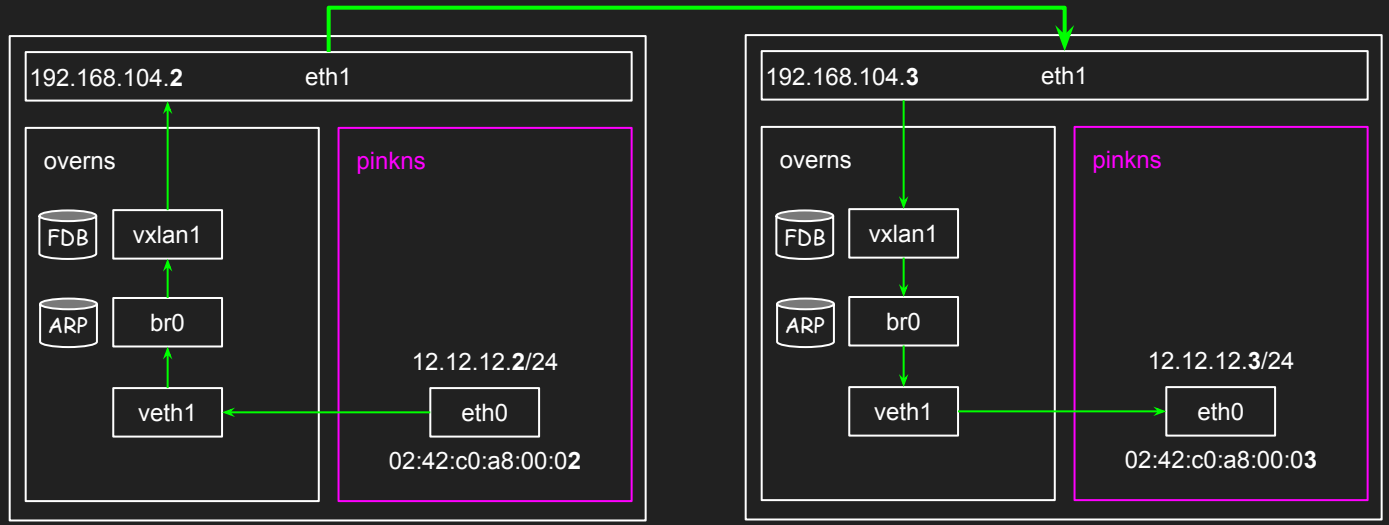
```
# ping 12.12.12.3
64 bytes from 12.12.12.3: icmp_seq=1 ttl=64 time=0.549 ms
```

터미널 #2 (overns@192.168.104.3)

```
# tcpdump -i vxlan1
…
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2933, seq 288, length 64
IP 12.12.12.3 > 12.12.12.2: ICMP echo reply, id 2933, seq 288, length 64
ARP, Request who-has 12.12.12.2 tell 12.12.12.3, length 28
ARP, Reply 12.12.12.2 is-at 02:42:c0:a8:00:02 (oui Unknown), length 28
…
```

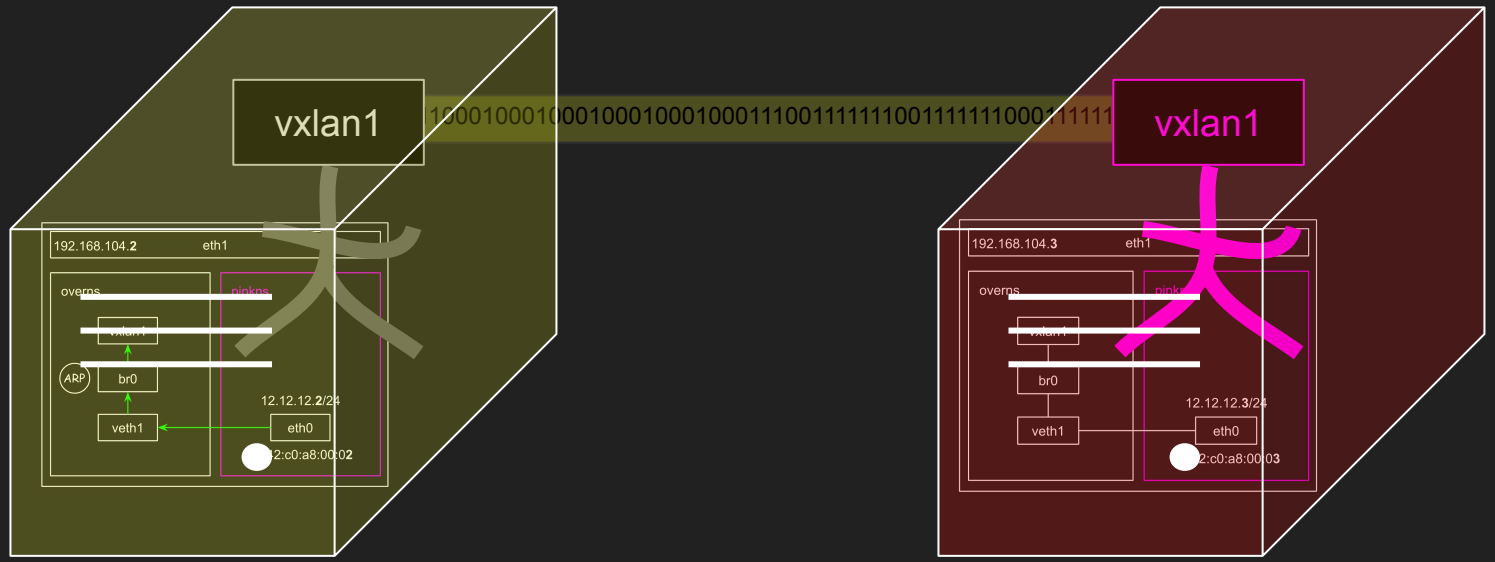
터미널 #3 (overns@192.168.104.2)

```
# tcpdump -i vxlan1
IP 12.12.12.2 > 12.12.12.3: ICMP echo request, id 2933, seq 3315 ...
IP 12.12.12.3 > 12.12.12.2: ICMP echo reply, id 2933, seq 3315 ...
```

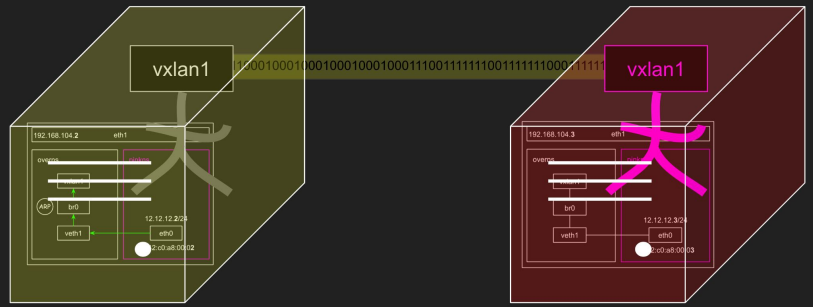


Overlay Network

*열일해준 **vxlan** 에게 박수 ~* 👏👏👏

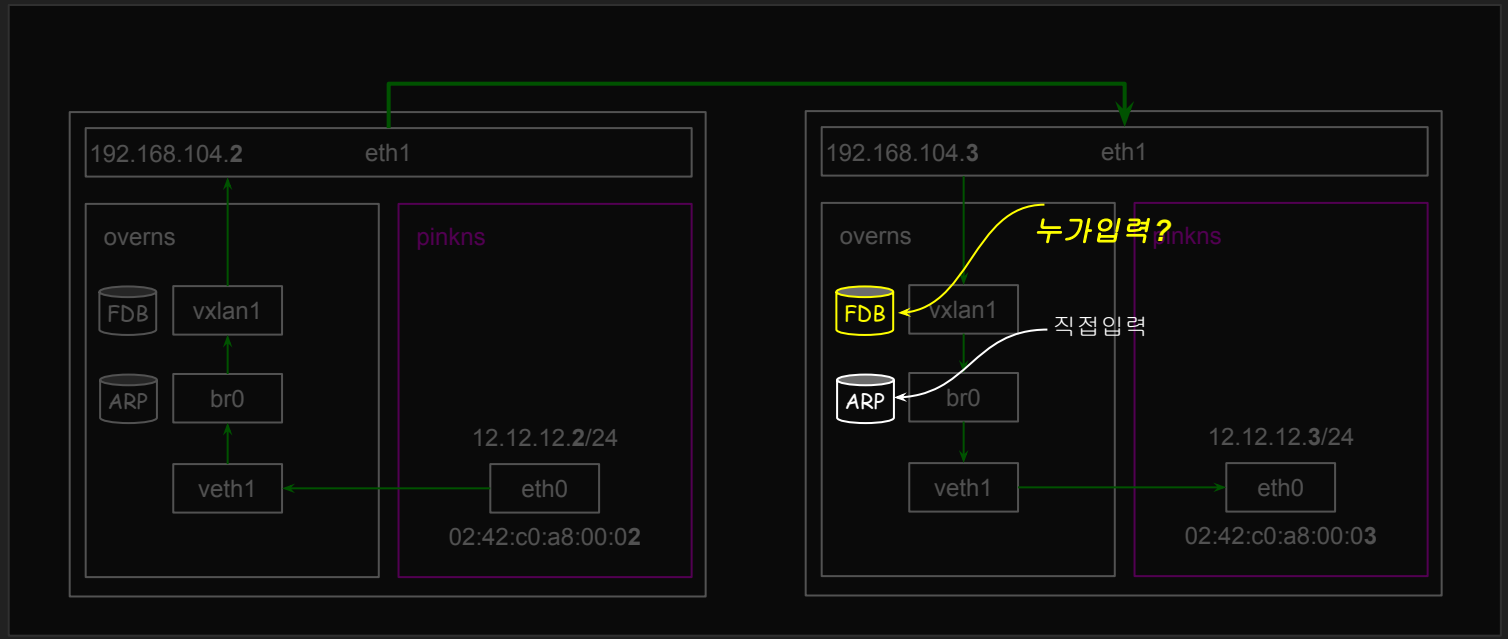


Overlay Network



그런데 말입니다 ...

bridge FDB는 어떻게 입력이 되었을까요?



그런데 말입니다 ...

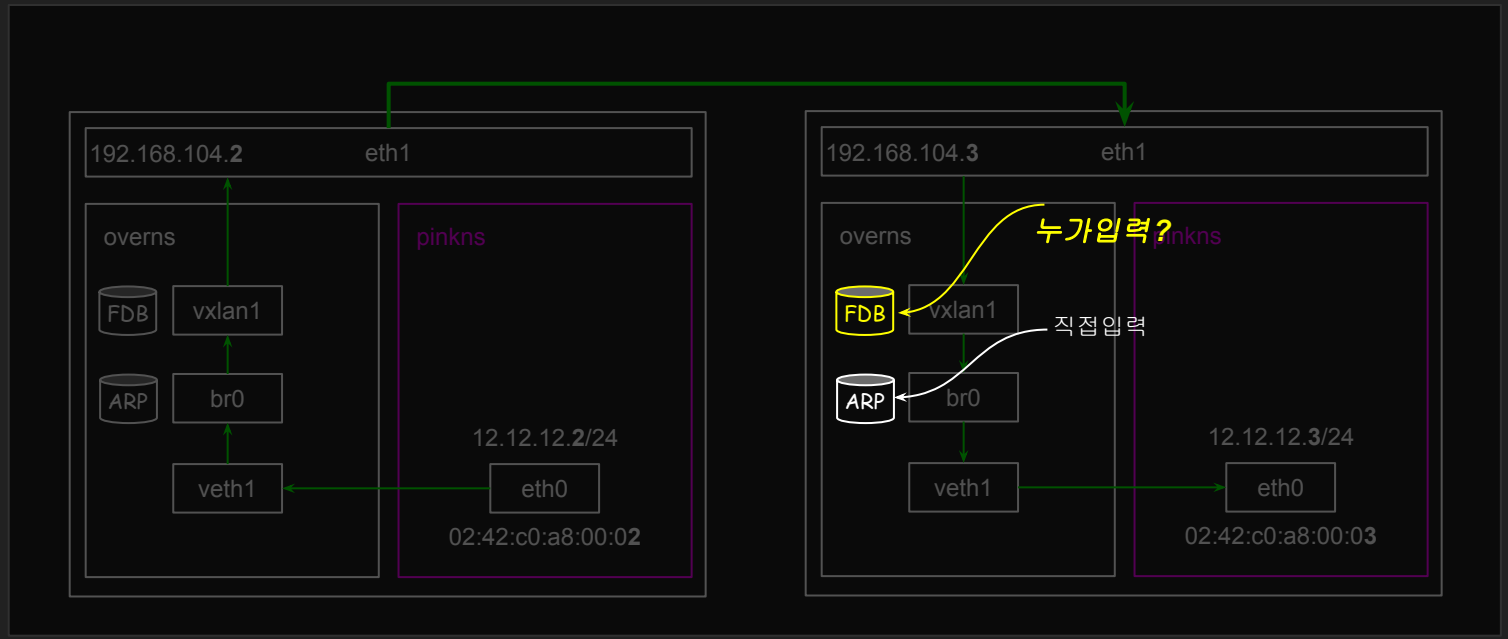
bridge FDB는 어떻게 입력이 되었을까요?

vxlan 디바이스 생성 시에 learning 옵션을 켜주었습니다

→ 커널이 요청 패킷을 검사하여 bridge fdb에 기록해 줍니다.

(지난 시간에 배운 bridge의 기능 중 포트별 MAC 주소를 기억하는 learning 이 있었죠)

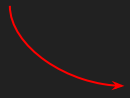
```
# ip link add dev vxlan1 type vxlan id 42 proxy learning dstport 4789
```



그런데 말입니다 ...

```
# 수동 입력
02:42:c0:a8:00:03 dev vxlan1 dst 192.168.104.3 link-netnsid 0 self permanent

# 자동 입력
02:42:c0:a8:00:02 dev vxlan1 dst 192.168.104.2 link-netnsid 0 self
```



***Aging** 에 의해서 일정시간이 지나면 삭제됨*

→ 지난 시간에 배운 … bridge는 aging 기능이 있고 TTL은 보통 300초

그런데 말입니다 ...

02:42:c0:a8:00:02 dev vxlan1 dst 192.168.104.2 link-netnsid 0 self

aging ? fdb에서 삭제되면 … 통신이 ???

네.. 12.12.12.3 → 12.12.12.2 으로의 통신은 되지 않습니다.

*12.12.12.2 로 부터 통신이 있지 않으면 **bridge fdb entry**가 갱신이 되지 않기 때문이죠*

bridge fdb 에서 02:42:c0:a8:00:02 를 지우고 ping 12.12.12.2 를 해보세요

터미널 #2 (overns@192.168.104.3)

```
# bridge fdb del 02:42:c0:a8:00:02 dev vxlan1 dst 192.168.104.2

# ping 12.12.12.2
. . .

--- 12.12.12.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1019ms
```

bridge fdb가 지워지면 통신이 되지 않습니다.

직접 fdb를 삭제 하긴 했습니다만.. 일정시간이 지나면 aging에 의하여 실제 fdb가 삭제됩니다

(실습1) 오버레이 네트워크

*192.168.104.3에서도 **bridge fdb**에 등록해 주세요*

터미널 #2 (overns@192.168.104.3)

VTEP ID
패킷을 수령할 엔드포인트

```
# bridge fdb replace 02:42:c0:a8:00:02 dev vxlan1 self dst 192.168.104.2 vni 42 port 4789
```

실제 목적지의 물리 IP 주소

실제 목적지의 물리 port

FDB

```
# bridge fdb show | grep 02:42:c0:a8:00:02
02:42:c0:a8:00:02 dev vxlan1 dst 192.168.104.2 link-netnsid 0 self permanent
```

이미 learning된 정보가 있으면 add는 실패하므로 replace 사용하였습니다

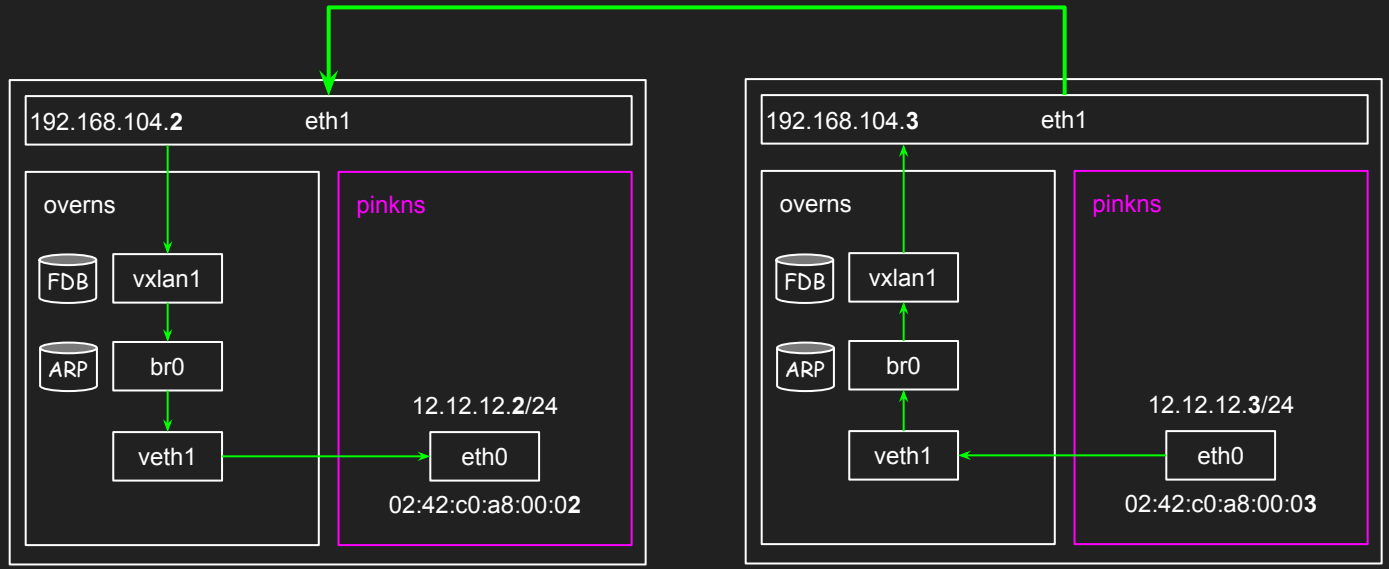
(실습1) 오버레이 네트워크

Ping 12.12.12.3 → 12.12.12.2

성공 ~

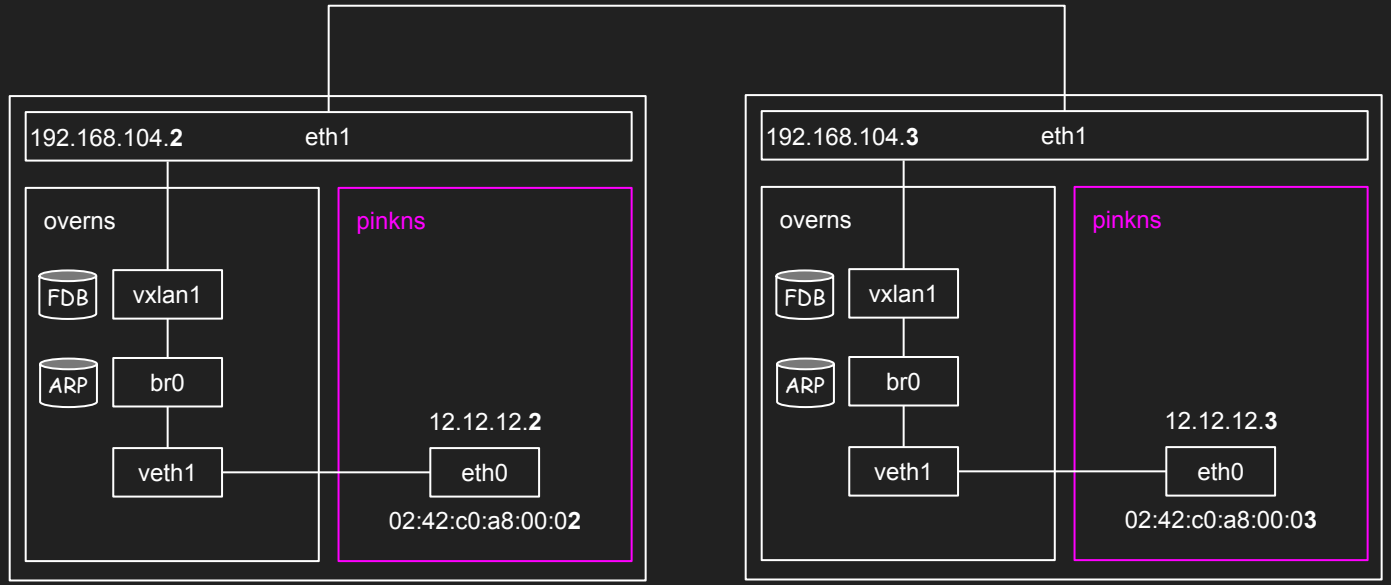
터미널 #2 (pinkns@192.168.104.3)

```
# ping 12.12.12.3 → 12.12.12.2
...
IP 12.12.12.3 > 12.12.12.2: ICMP echo request, id 3933, seq 288, length 64
IP 12.12.12.2 > 12.12.12.3: ICMP echo reply, id 3933, seq 288, length 64
...
```



(실습1) 오버레이 네트워크

그림 완성



목차 보기

END