실습과 함께 완성해보는

도커 없이 컨테이너 만들기

3편

Sam.0

시작하기에 앞서 ...

본 컨텐츠는 앞편을 학습하였다고 가정하고 준비되었습니다.

원활한 이해 및 실습을 위하여 앞편을 먼저 보시기를 추천드립니다

1편 링크 클릭 *2편 링크 클릭*

실습을 위한 사전 준비 사항

실습은 . . .

- 맥 환경에서 VirtualBox + Vagrant 기반으로 준비되었습니다
  - 맥 이외의 OS 환경도 괜찮습니다만
  - 원활한 실습을 위해서
  - “VirtualBox or VMware + Vagrant”는 권장드립니다.
- 실습환경 구성을 위한 Vagrantfile을 제공합니다.

실습을 위한 사전 준비 사항

Vagrantfile

- 오른쪽의 텍스트를 복사하신 후
- 로컬에 Vagrantfile로 저장하여 사용하면 됩니다.
- vagrant 사용법은 공식문서를 참고해 주세요 https://www.vagrantup.com/docs/index

```
BOX_IMAGE = "bento/ubuntu-18.04"
HOST_NAME = "ubuntu1804"

$pre_install = <<-SCRIPT
  echo ">>>> pre-install <<<<<<"
  sudo apt-get update &&
  sudo apt-get -y install gcc &&
  sudo apt-get -y install make &&
  sudo apt-get -y install pkg-config &&
  sudo apt-get -y install libseccomp-dev &&
  sudo apt-get -y install tree &&
  sudo apt-get -y install jq &&
  sudo apt-get -y install bridge-utils

  echo ">>>> install go <<<<<<"
  curl -O https://storage.googleapis.com/golang/go1.15.7.linux-amd64.tar.gz > /dev/null 2>&1 &&
  tar xf go1.15.7.linux-amd64.tar.gz &&
  sudo mv go /usr/local/ &&
  echo 'PATH=$PATH:/usr/local/go/bin' | tee /home/vagrant/.bash_profile

  echo ">>>>> install docker <<<<<<"
  sudo apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common > /dev/null 2>&1 &&
  sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - &&
  sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" &&
  sudo apt-get update &&
  sudo apt-get -y install docker-ce docker-ce-cli containerd.io > /dev/null 2>&1
SCRIPT

Vagrant.configure("2") do |config|

 config.vm.define HOST_NAME do |subconfig|
   subconfig.vm.box = BOX_IMAGE
   subconfig.vm.hostname = HOST_NAME
   subconfig.vm.network :private_network, ip: "192.168.104.2"
   subconfig.vm.provider "virtualbox" do |v|
     v.memory = 1536
     v.cpus = 2
   end
   subconfig.vm.provision "shell", inline: $pre_install
 end

end
```

실습을 위한 사전 준비 사항

Vagrantfile 제공 환경

vagrant + virtual vm
ubuntu 18.04
docker 20.10.5 ** 도커 이미지 다운로드 및 컨테이너 비교를 위한 용도로 사용합니다.*

기타 설치된 툴
~ tree, jq, brctl, … 등 실습을 위한 툴

실습 계정 (root)

```
# sudo -Es
```

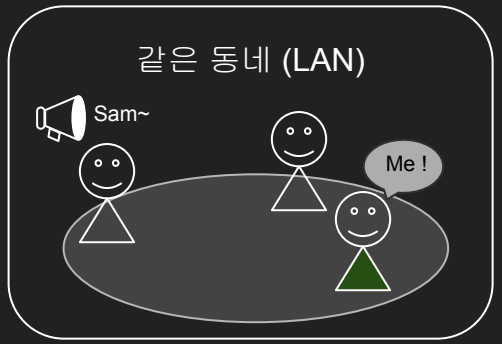
실습 폴더

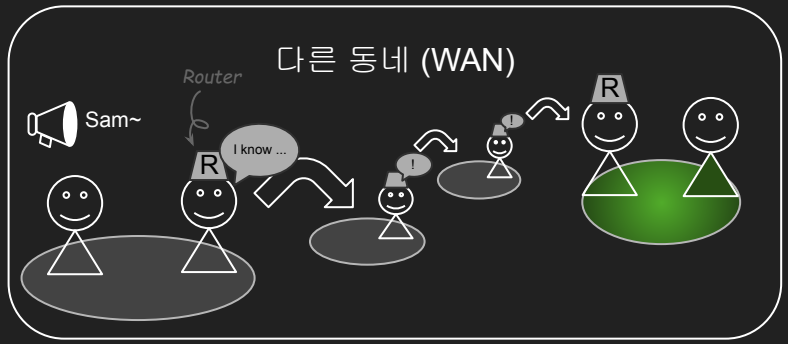
```
# cd /tmp
```

Network ?

연결

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





Network ?

통신

- 유니캐스트 1:1
- 브로드캐스트 ALL
- 멀티캐스트 1:N

눈치껏 보내고 충돌(collision) 나면 retry

이더넷 (ethernet) ? 컴퓨터 네트워킹의 한 방식

- CSMA/CD (눈치게임)
  - Carrier Sense *신호감지*
  - Multiple Access *다중접근*
  - Collision Detection *충돌감지*

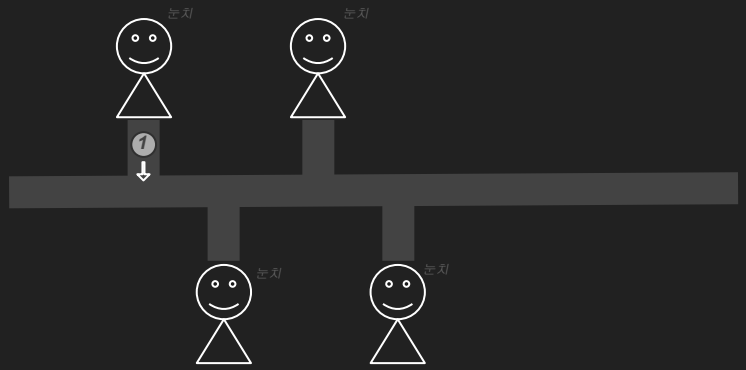
이미지출처:
https://www.youtube.com/watch?v=U2phvIP3Beo

이더넷

눈치껏 보내고 충돌(collision) 나면 retry

이더넷 (ethernet) ? 컴퓨터 네트워킹의 한 방식

- CSMA/CD (눈치게임)
  - Carrier Sense *신호감지*
  - Multiple Access *다중접근*
  - Collision Detection *충돌감지*



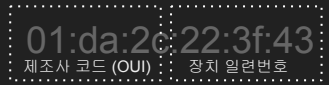
담당 { *MAC 프레임/프로토콜* / *신호/배선*

출처: 위키 - 이더넷

MAC address

MAC (Media Access Control) Address

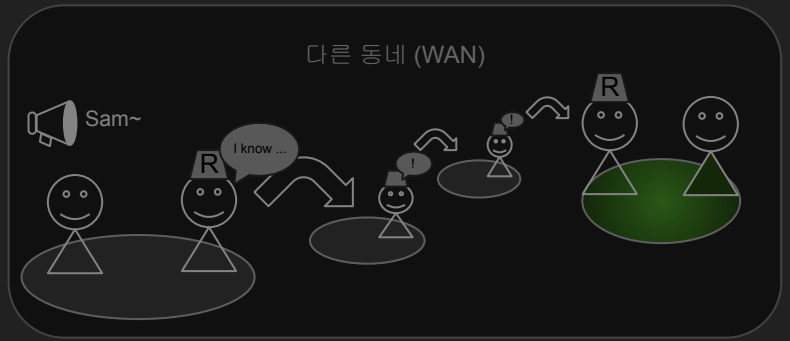
- Physical Address
- 6옥텟
- ARP



ARP

ARP (Address Resolution Protocol)

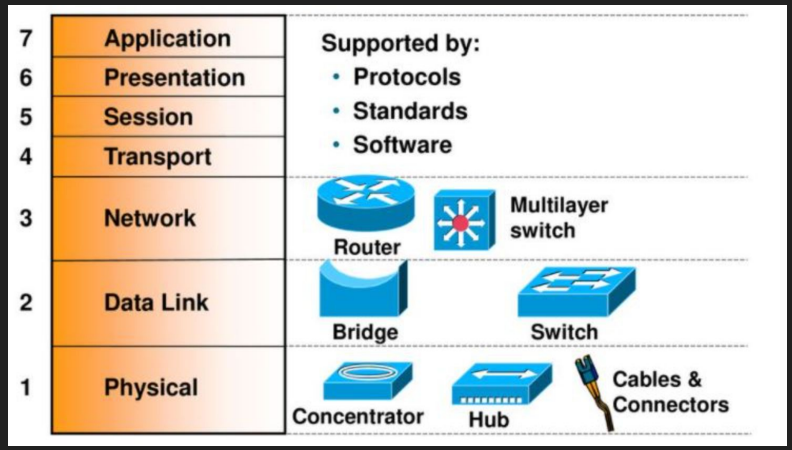
- 주소(MAC) 를 물어보는 프로토콜 예) 11.11.11.3의 MAC 주소가 뭐에요?
- 브로드캐스트 (FF:FF:FF:FF:FF:FF)
- 라우터는 본인의 MAC주소를 알려줌 “나한테 물어봐 Sam이 어디사는지 알려줄게”



OSI 7 Layer

OSI 7 Layer 가 만들어지게 된 배경

~ 상호 이질적인 네트워크 간의 연결 호환성 필요
~ 통신장비업체들이 자사 장비끼리만 호환되게 제작

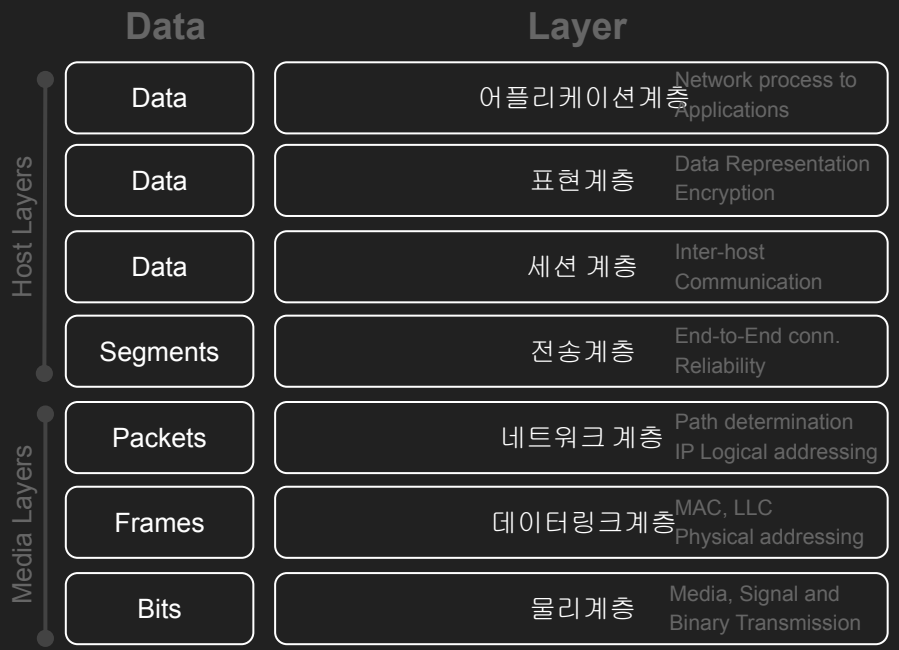


참고:
OSI 7 Layer http://blog.naver.com/demonicws/40117378644
OSI 7 Layer 계층별 장비https://togll.tistory.com/40

OSI 7 Layer

OSI (Open System Interconnection) 7 Layer

- 왜 만들었지?
  - 층별 “표준화” (호환성 확보) ~ 비용절감
  - 데이터의 흐름 파악 용이 ~ 학습도구로 활용
  - 문제진단/해결 용이
  - 통신이 편리

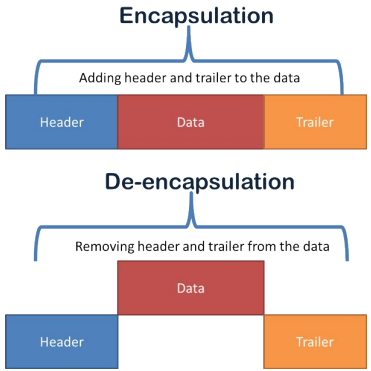


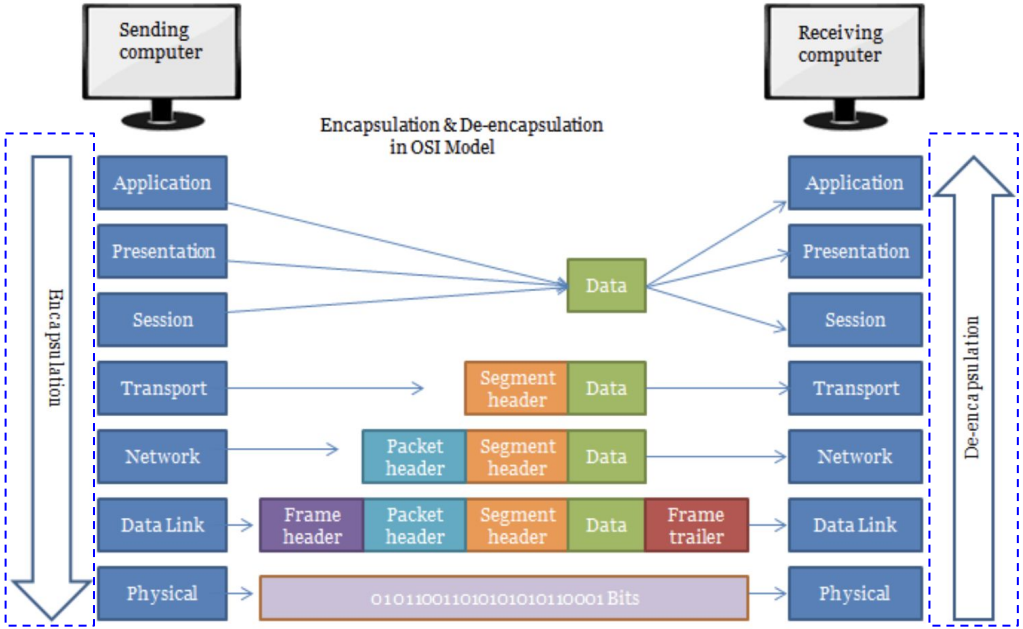
참고: OSI 7 Layer :http://blog.naver.com/demonicws/40117378644

어떻게 레이어 구조를 만들었을까요?

Network Stack

- 캡슐화 : Layer 구분 / Layer간 통신





출처 링크: [Data Encapsulation and De-encapsulation Explained]

지금까지 네트워크 전반에 대하여 간략히 다뤄보았습니다

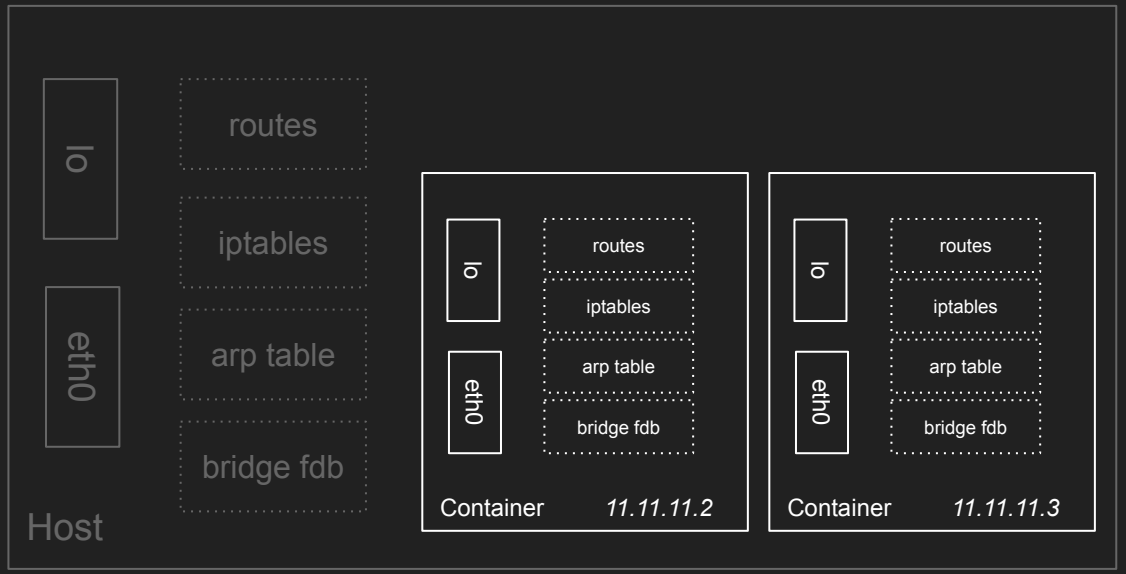
본격적으로 네트워크 네임스페이스에 대해 이야기 해봅시다

Network namespace

호스트 안의 또 다른 가상 네트워크

Isolates network and virtualize network stack

네트워크를 isolation하고 네트워크 스택(OSI 7L)을 가상화



IP주소, 라우팅테이블, 소켓, 방화벽, … 등 모든게 별도로 가상화 네트워크에 준비됩니다

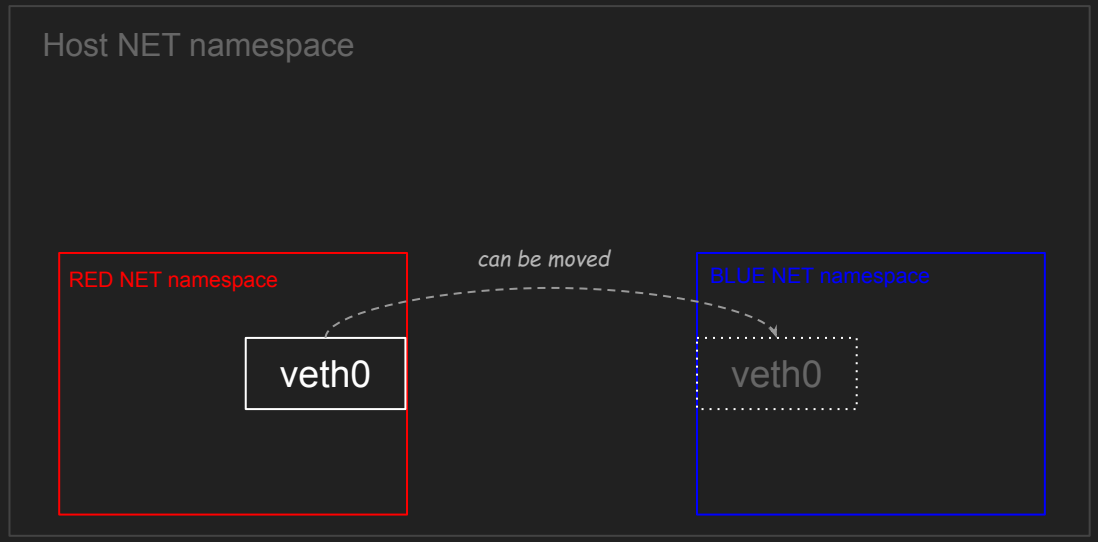
Network namespace

호스트 안의 또 다른 가상 네트워크

Network interface
- is present in exactly 1 namespace
- can be moved between namespaces

Network interface는 랜카드, 브릿지 같은 장치를 생각하시면 됩니다

가상 디바이스를 생성하고 물리장치를 다루듯이
서로 다른 네트워크 네임스페이스 간에 옮겨 꼽을 수 있습니다

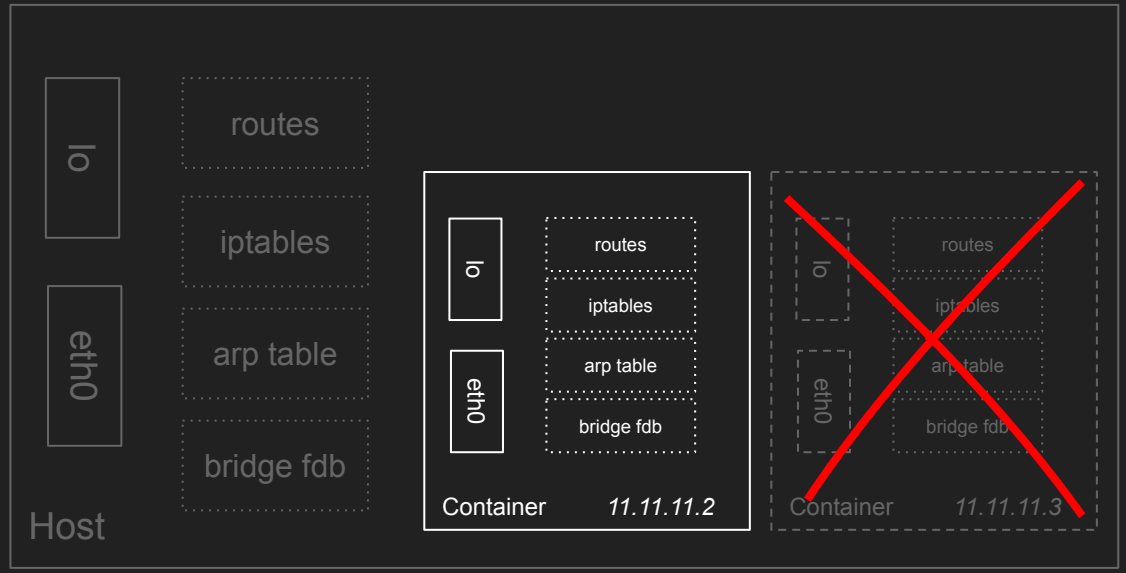


Network namespace

호스트 안의 또 다른 가상 네트워크

Destroying a network namespace
- virtual interfaces : destroyed
- physical interfaces : back to the initial network namespace

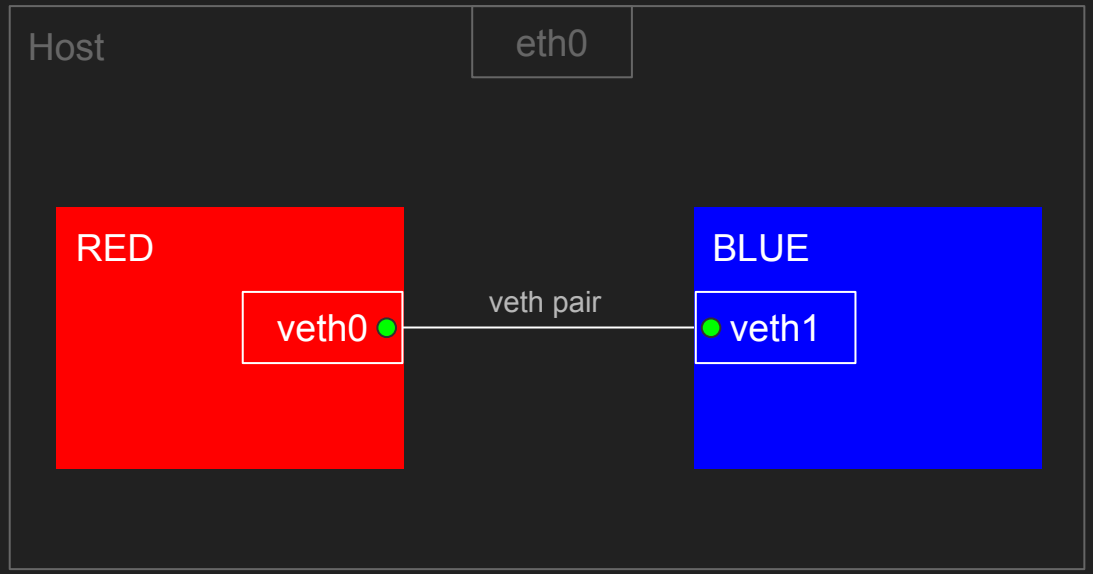
네트워크 네임스페이스를 삭제하면 포함된 모든 것들이 삭제됩니다
~ one shot one kill (지우기 편합니다 ㅎㅎ)



(실습1) 네임스페이스 간 1:1 통신

지금부터 그릴 그림입니다

RED / BLUE 간 통신을 구현해 봅시다



(실습1) 네임스페이스 간 1:1 통신

veth (virtual ethernet) 는 랜카드를 상상하시면 됩니다

```
# ip link add veth0 type veth peer name veth1
```

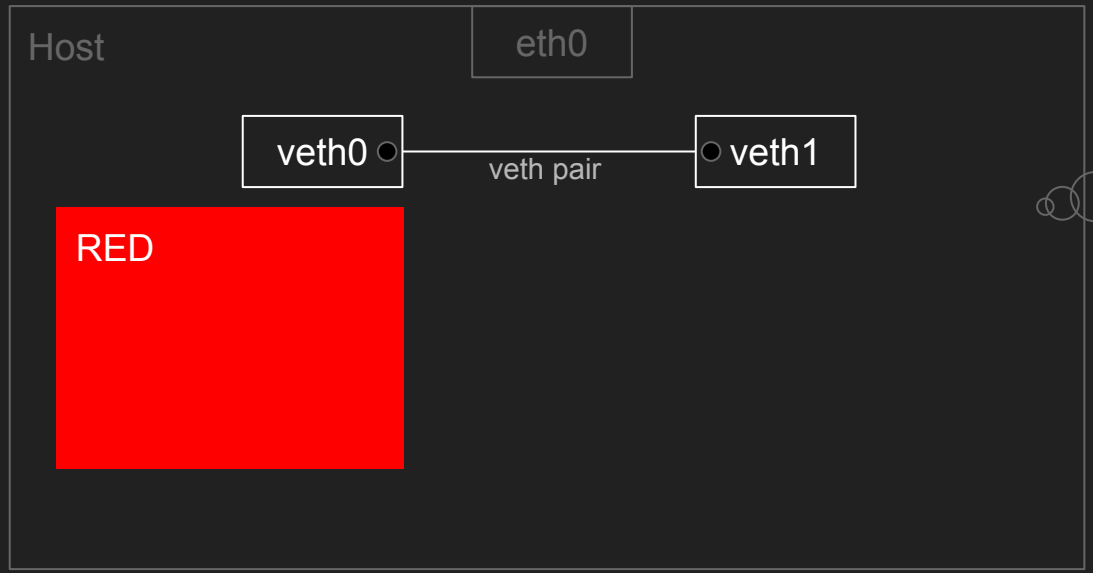

```
IP-LINK(8)                                    Linux

NAME
       ip-link - network device configuration
```

※ ip 명령어는 네트워크 관련해서는 앞으로 굉장히 자주 사용하게 될 명령어인데요
man 페이지를 참고해 주세요https://man7.org/linux/man-pages/man8/ip.8.html

(실습1) 네임스페이스 간 1:1 통신

네트워크 네임스페이스(RED)를 생성합니다

```
# ip netns add RED
```



빨간색 본체를 하나 준비 하였다고 생각해 보세요

```
IP-NETNS(8)                                    Linux

NAME
       ip-netns - process network namespace management
```

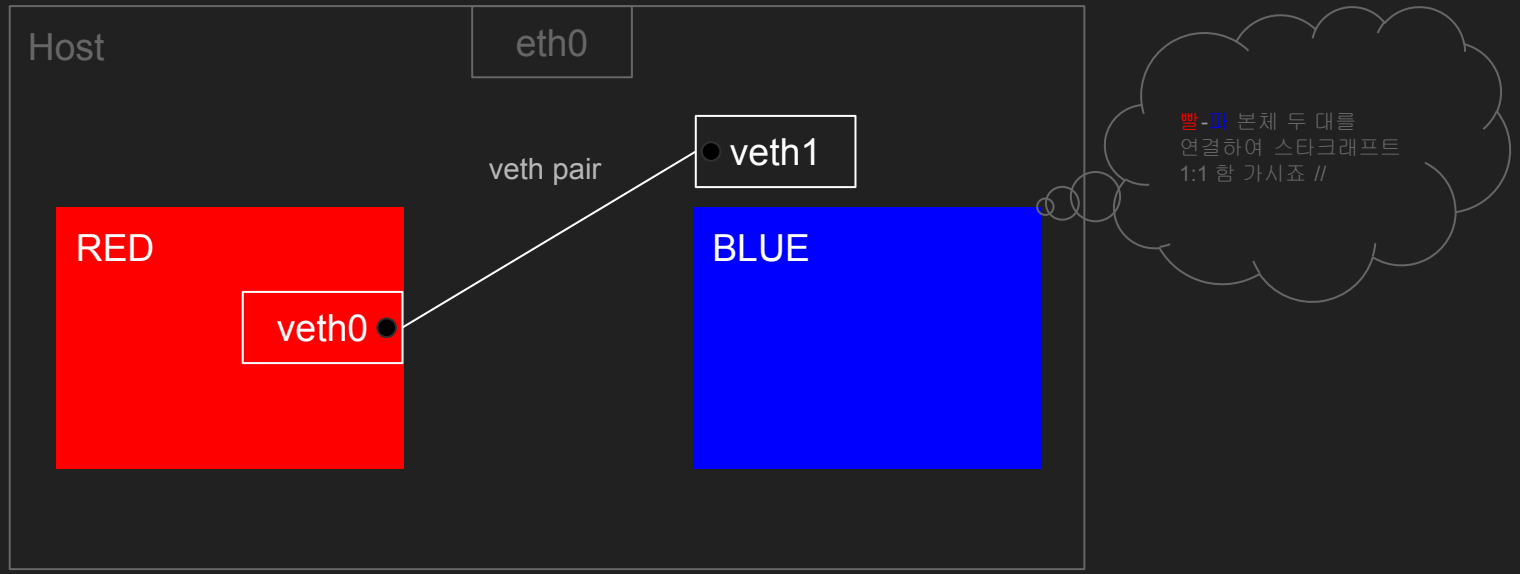
(실습1) 네임스페이스 간 1:1 통신

네트워크 네임스페이스(BLUE)를 생성합니다

```
# ip netns add BLUE
```

(실습1) 네임스페이스 간 1:1 통신

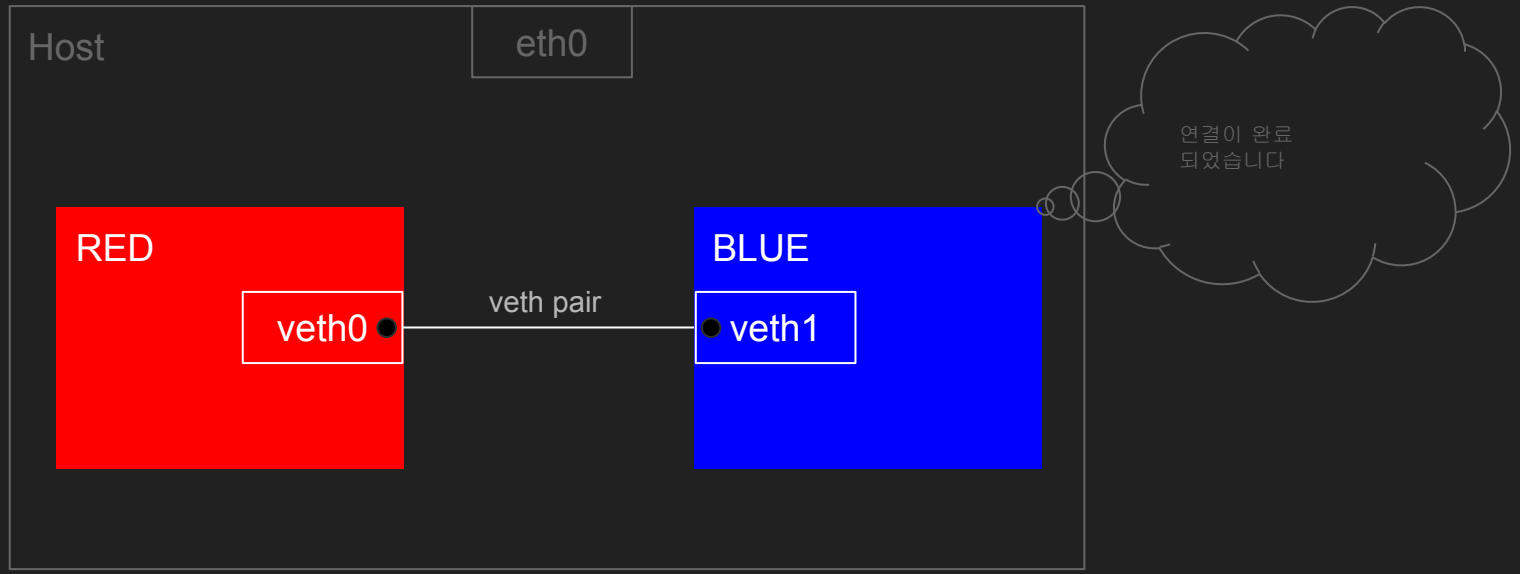
```
# ip link set veth0 netns RED
```



"...Network interface can be moved between namespaces"

(실습1) 네임스페이스 간 1:1 통신

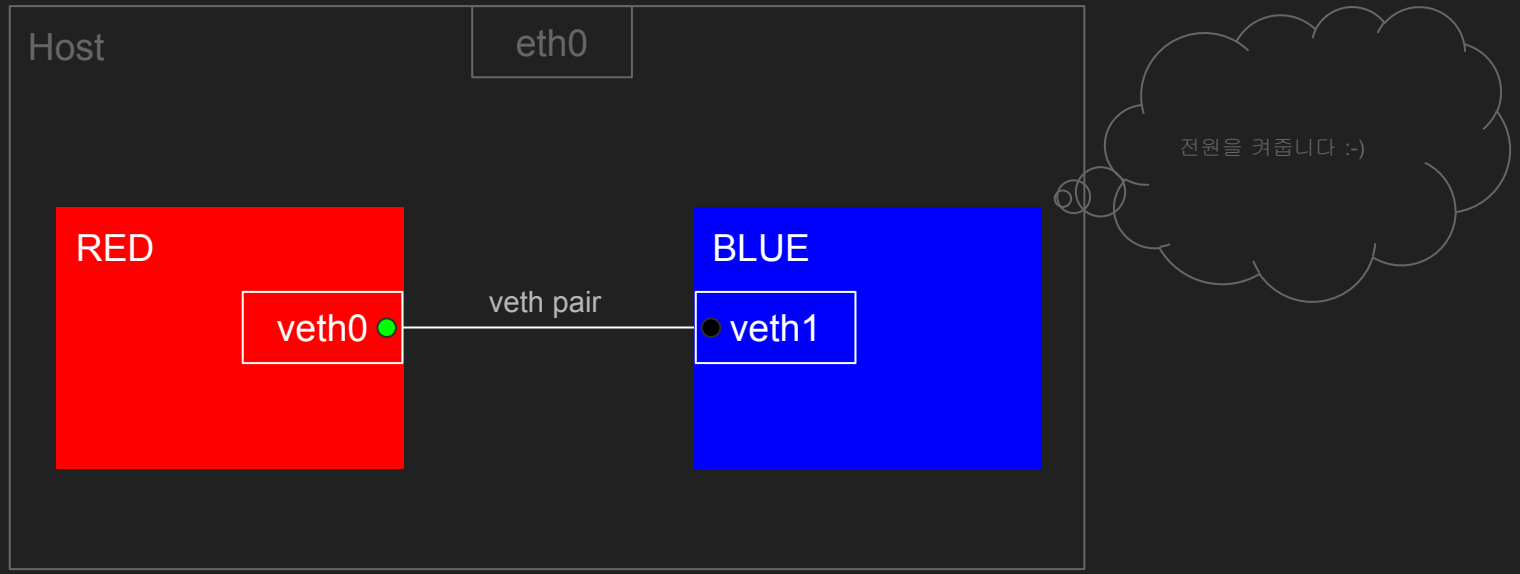
```
# ip link set veth1 netns BLUE
```



"...Network interface can be moved between namespaces"

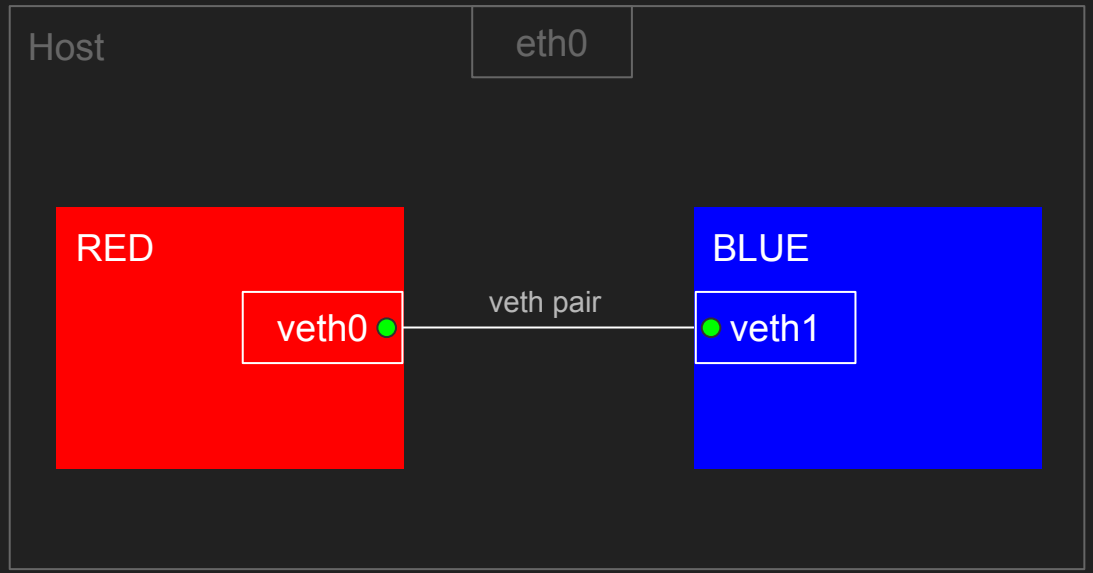
(실습1) 네임스페이스 간 1:1 통신

```
# ip netns exec RED ip link set veth0 up
```



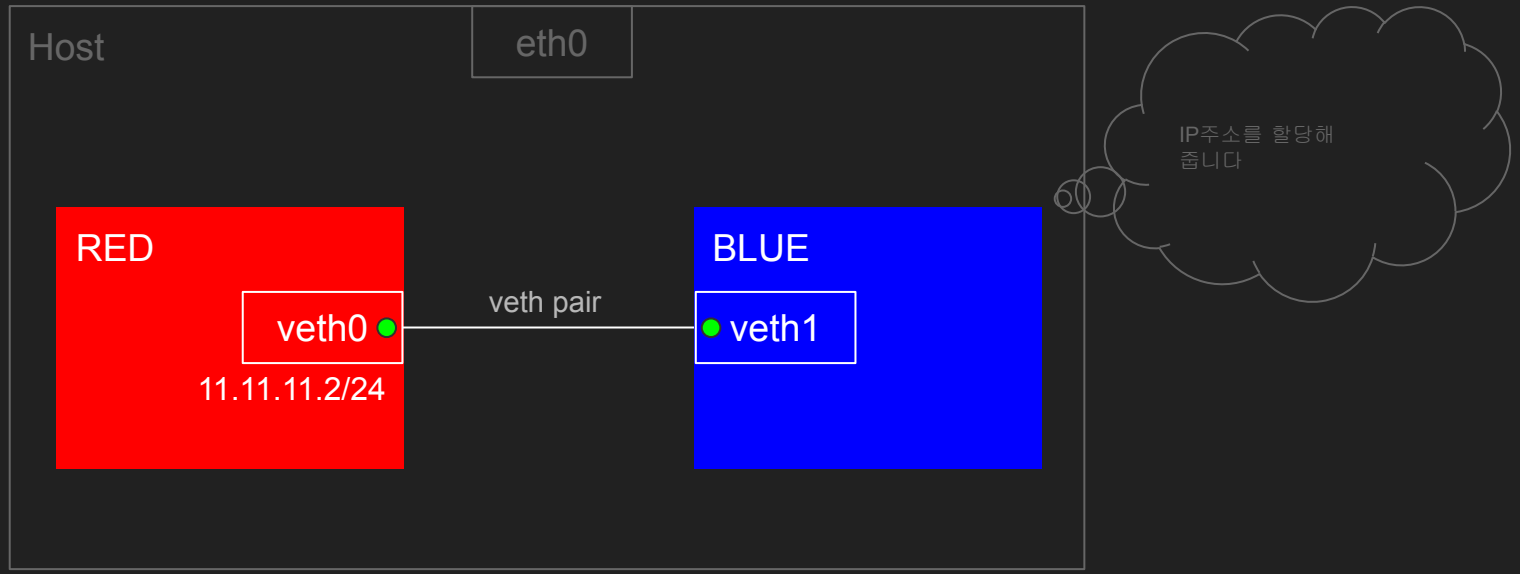
(실습1) 네임스페이스 간 1:1 통신

```
# ip netns exec BLUE ip link set veth1 up
```



(실습1) 네임스페이스 간 1:1 통신

```
# ip netns exec RED ip addr add 11.11.11.2/24 dev veth0
```

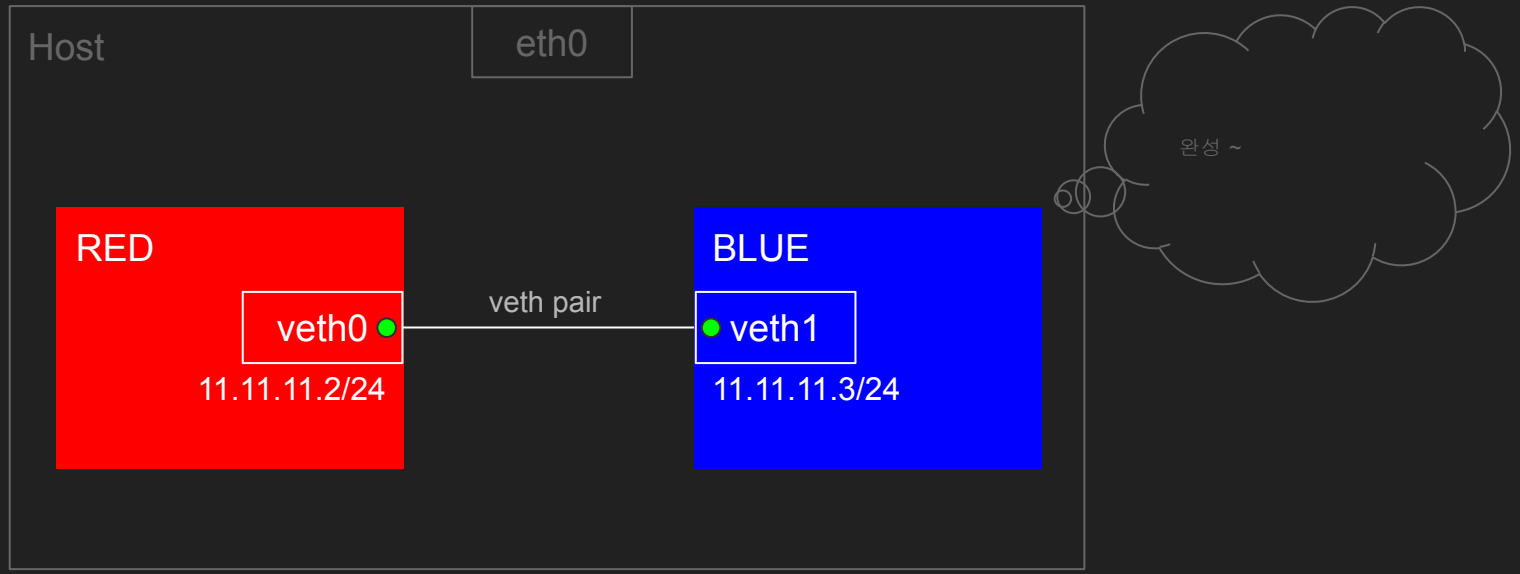


ip address ~ protocol address management

- *줄여서 **ip addr** 로 많이씀*

(실습1) 네임스페이스 간 1:1 통신

```
# ip netns exec BLUE ip addr add 11.11.11.3/24 dev veth1
```



ip address add ~ add new protocol address

- *dev IFNAME : 주소를 설정할 장치명*

(실습1) 네임스페이스 간 1:1 통신

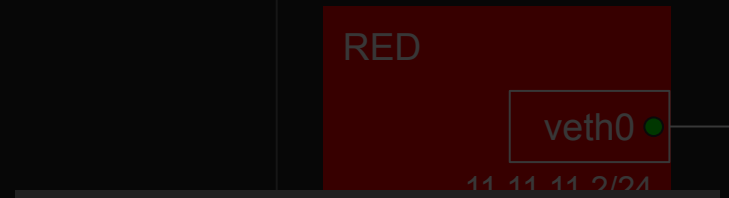
호스트와 RED를 비교해 봅시다

터미널 #1 (RED, 11.11.11.2)

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

터미널 #2 (호스트)

```
#
```



veth pair

터미널 (#2)창을 하나더 열어서 실습환경(VM)으로 접속해주세요

Vagrantfile 이 있는 위치에서

$ vagrant ssh ubuntu1804

nsenter [OPTION] [COMMAND]

네임스페이스에 attach하여 지정한 프로그램을 실행합니다

COMMAND : 지정하지 않으면(default), $SHELL을 실행

--net : Net namespace

~ 옵션으로 target namespace type을 지정합니다

예) : --mount , --pid, --ipc, --user , …

2편 참고

참고) # man nsenter

(실습1) 네임스페이스 간 1:1 통신

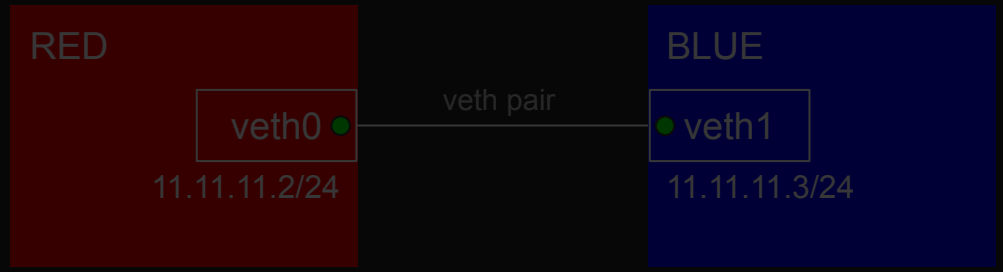
호스트와 RED를 비교해 봅시다

터미널 #1 (RED, 11.11.11.2)

```
# ip address show
```

터미널 #2 (호스트)

```
# ip address show
```



ip address show ~ 각 네임스페이스의 네트워크 장치를 확인합니다

- *줄여서 ip addr show*
- *혹은 ip addr (show가 default)*

(실습1) 네임스페이스 간 1:1 통신

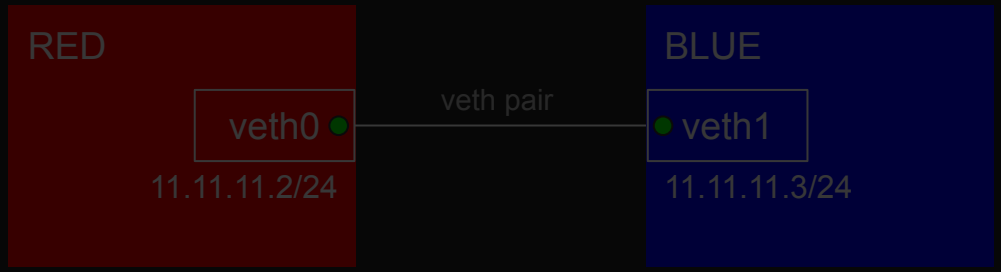
호스트와 RED를 비교해 봅시다

터미널 #1 (RED, 11.11.11.2)

```
# ip neighbour show
```

터미널 #2 (호스트)

```
# ip neighbour show
```



ip-neighbour neighbour/arp tables management *참고) # man ip-neighbour*

- 줄여서 "ip neigh" 로 많이 씀
- ip neigh show
- ip neigh
  - 뒤에 동작을 안주면 default 는 show 입니다

(실습1) 네임스페이스 간 1:1 통신

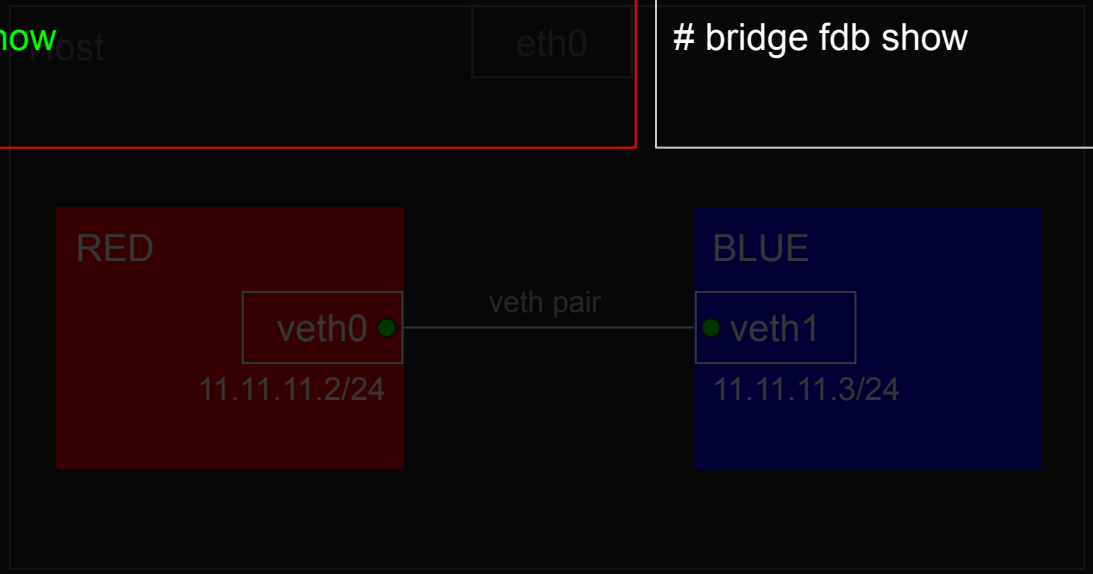
호스트와 RED를 비교해 봅시다

터미널 #1 (RED, 11.11.11.2)

```
# bridge fdb show
```

터미널 #2 (호스트)

```
# bridge fdb show
```



(실습1) 네임스페이스 간 1:1 통신

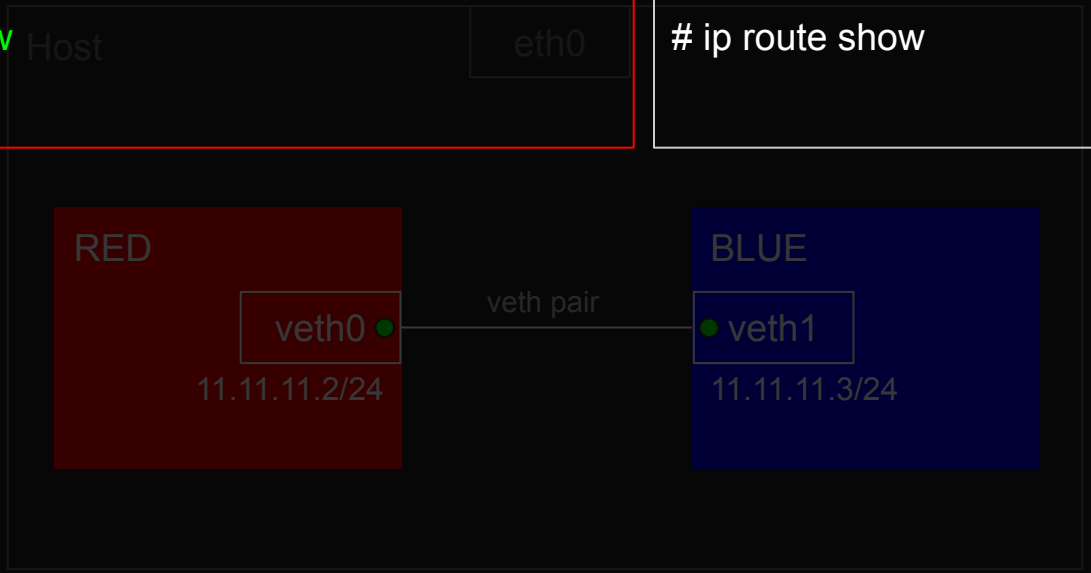
호스트와 RED를 비교해 봅시다

터미널 #1 (RED, 11.11.11.2)

```
# ip route show
```

터미널 #2 (호스트)

```
# ip route show
```



(실습1) 네임스페이스 간 1:1 통신

호스트와 RED를 비교해 봅시다

터미널 #1 (RED, 11.11.11.2)

```
# iptables -L
```

터미널 #2 (호스트)

```
# iptables -L
```

(실습1) 네임스페이스 간 1:1 통신

터미널 #1 (RED, 11.11.11.2)

```
#
```

터미널 #2 (BLUE, 11.11.11.3)

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

(실습1) 네임스페이스 간 1:1 통신

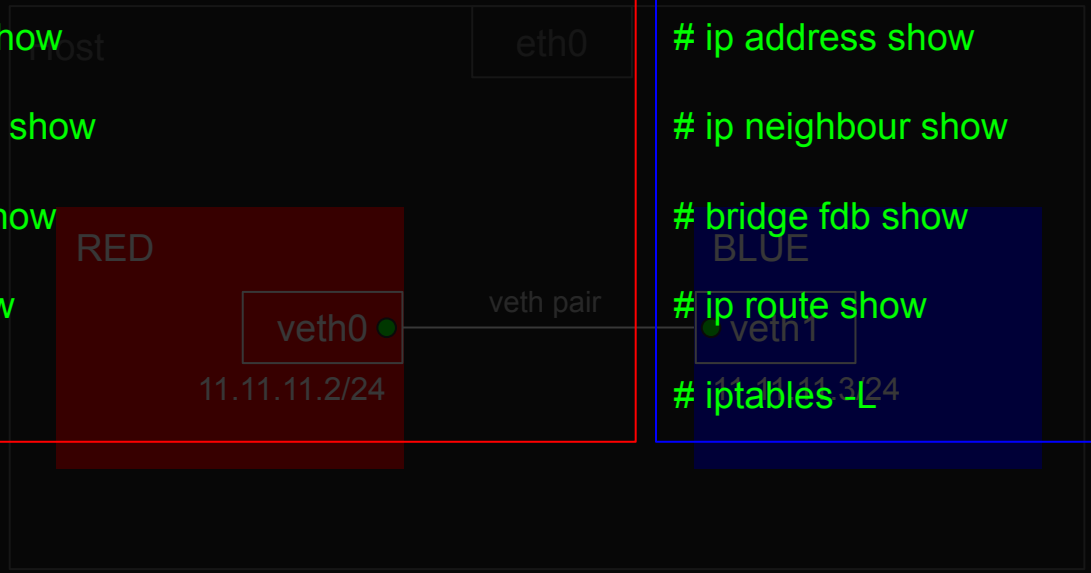
RED와 BLUE를 비교해 보세요

터미널 #1 (RED, 11.11.11.2)

```
# ip address show
# ip neighbour show
# bridge fdb show
# ip route show
# iptables -L
```

터미널 #2 (BLUE, 11.11.11.3)

```
# ip address show
# ip neighbour show
# bridge fdb show
# ip route show
# iptables -L
```



(실습1) 네임스페이스 간 1:1 통신

이웃 정보를 확인해 둡니다 (아직 아무 이웃이 없네요)

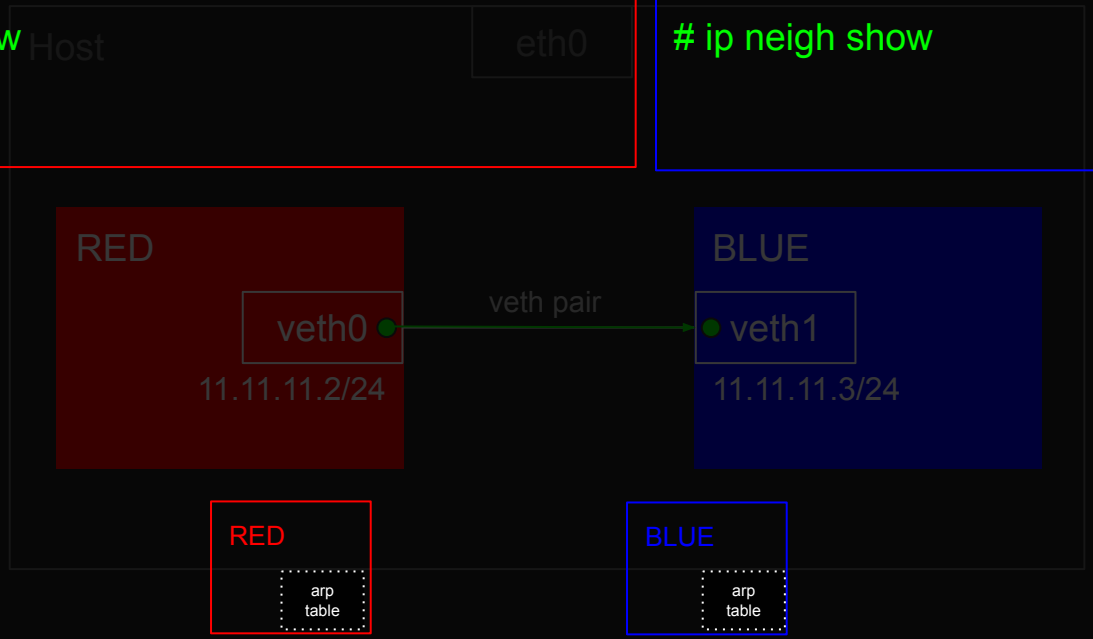
ip netns exec BLUE ping -c 3 11.11.11.2

터미널 #1 (RED, 11.11.11.2)

```
# ip neigh show
```

터미널 #2 (BLUE, 11.11.11.3)

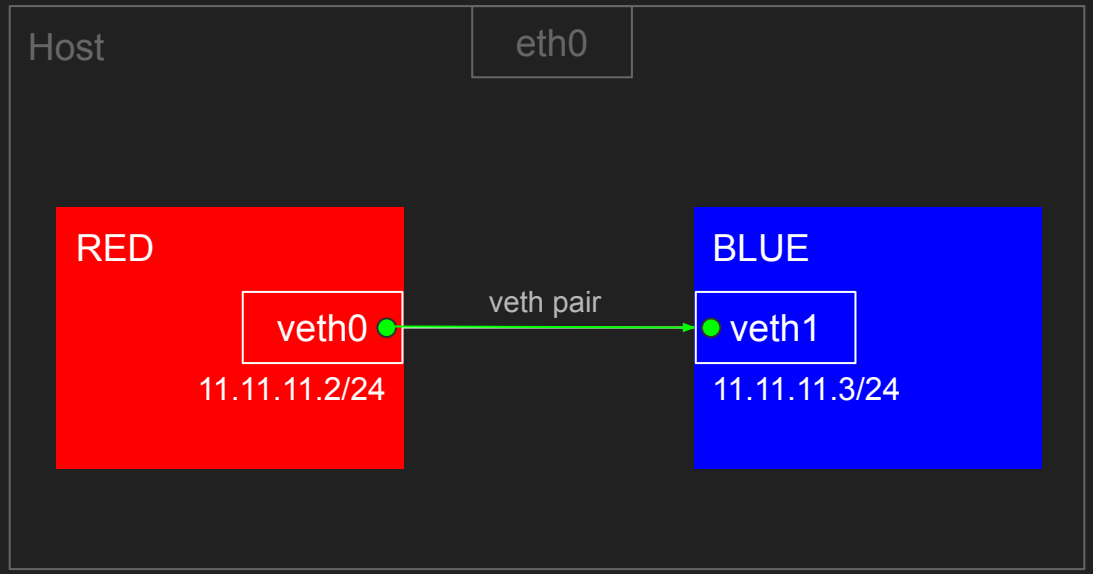
```
# ip neigh show
```



(실습1) 네임스페이스 간 1:1 통신

RED → BLUE

Ping 을 한번 보내 보겠습니다



(실습1) 네임스페이스 간 1:1 통신

PING을 보내면 먼저 서로 ARP정보를 확인합니다

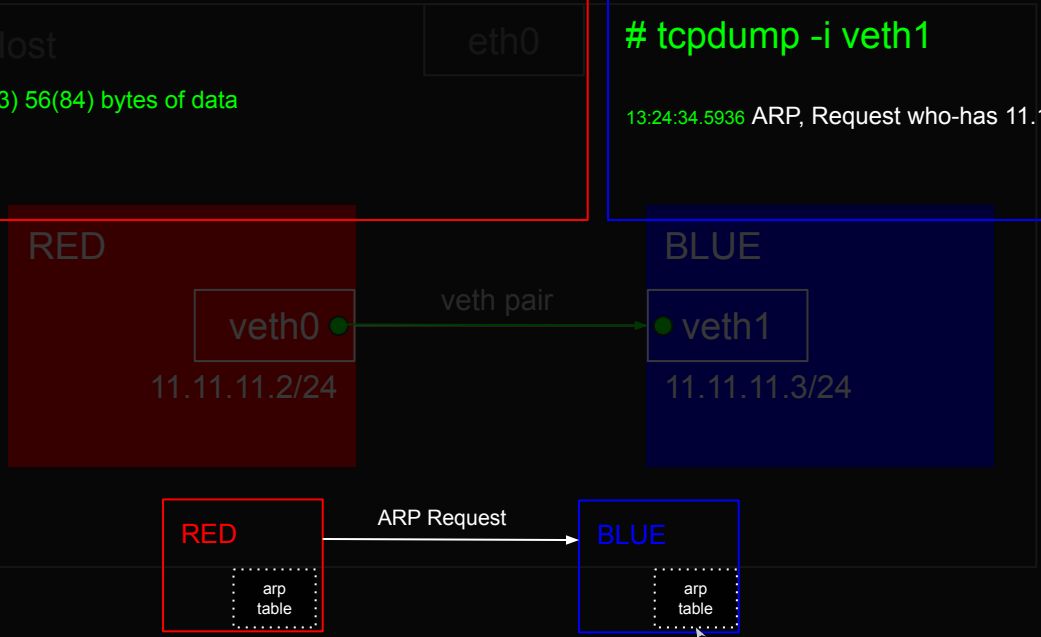
ip netns exec BLUE ping -c 3 11.11.11.2

터미널 #1 (RED, 11.11.11.2)

```
# ping 11.11.11.3
PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data
```

터미널 #2 (BLUE, 11.11.11.3)

```
# tcpdump -i veth1
13:24:34.5936 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
```



(실습1) 네임스페이스 간 1:1 통신

상대편 ARP정보를 arp table에 기록합니다

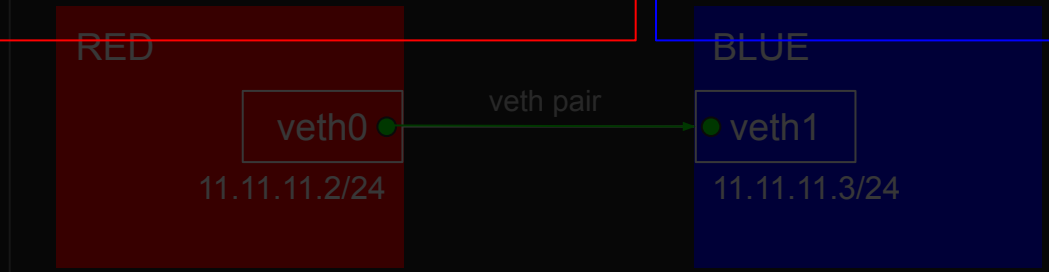
ip netns exec BLUE ping -c 3 11.11.11.2

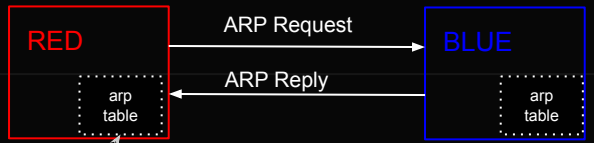
터미널 #1 (RED, 11.11.11.2)

```
# ping -c 1 11.11.11.3

PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data
```

터미널 #2 (BLUE, 11.11.11.3)

```
# tcpdump -i veth1

13:24:34.5936 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
13:24:34.5937 ARP, Reply 11.11.11.2 is-at 1e:12:5b:e9:80:ca (oui Unknown), length 28
```





11.11.11.3 dev reth0 lladdr 1e:12:5b:e9:80:ca STALE

11.11.11.2 dev reth0 lladdr 8a:2c:bd:a5:6a:ff STALE

(실습1) 네임스페이스 간 1:1 통신

그리고 난 뒤에 실제 PING 패킷(ICMP) 을 주고 받습니다

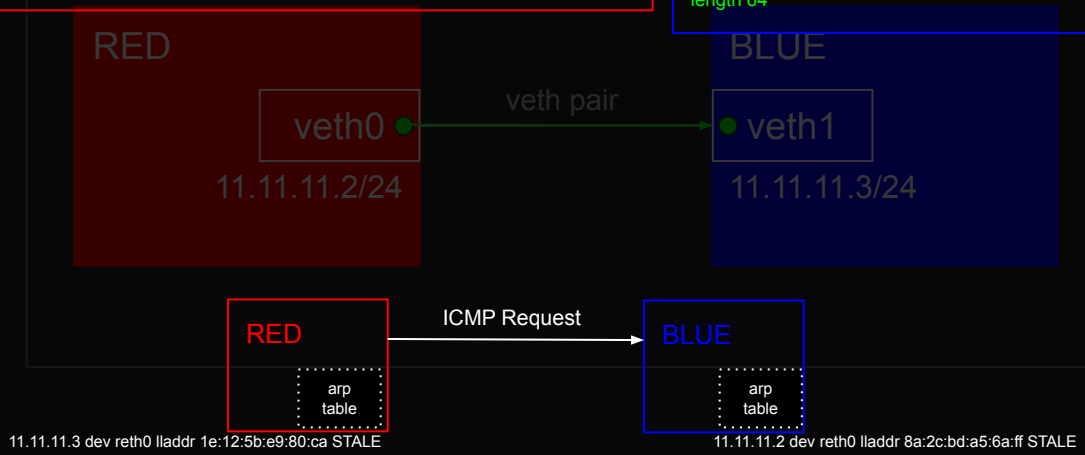
ip netns exec BLUE ping -c 3 11.11.11.2

터미널 #1 (RED, 11.11.11.2)

```
# ping 11.11.11.3

PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data
```

터미널 #2 (BLUE, 11.11.11.3)

```
# tcpdump -i veth1

13:24:34.5936 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
13:24:34.5937 ARP, Reply 11.11.11.2 is-at 1e:12:5b:e9:80:ca (oui Unknown), length 28
13:24:34.9780 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 31132, seq 1, length 64
```



(실습1) 네임스페이스 간 1:1 통신

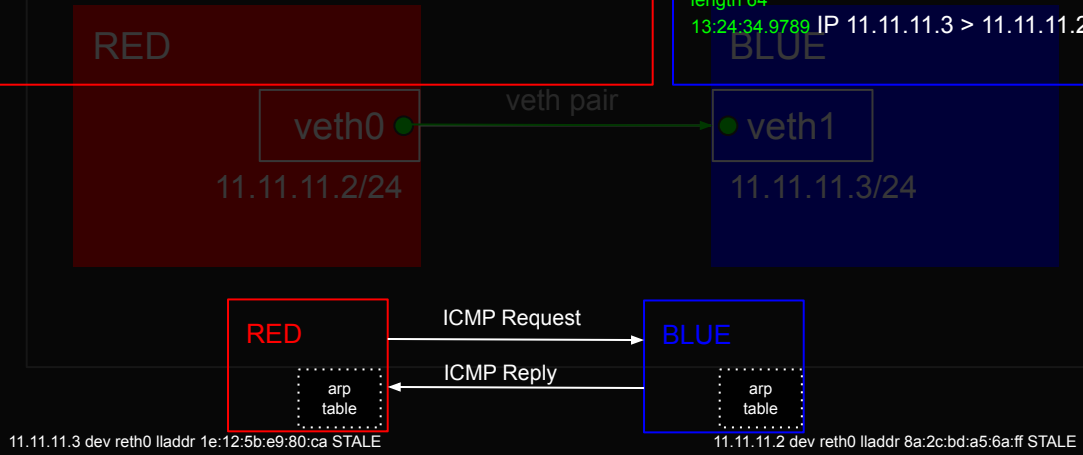
RED/BLUE 통신 성공

터미널 #1 (RED, 11.11.11.2)

```
# ping 11.11.11.3

PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data
64 bytes from 11.11.11.3: icmp_seq=17 ttl=64 time=0.038 ms
```

터미널 #2 (BLUE, 11.11.11.3)

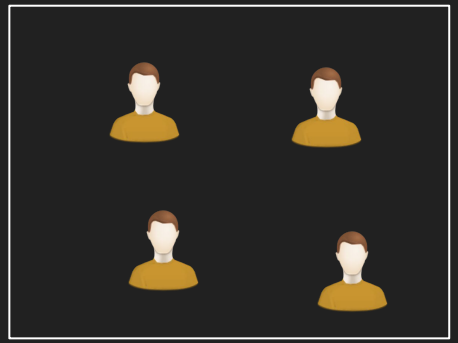
```
# tcpdump -i veth1

13:24:34.5936 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
13:24:34.5937 ARP, Reply 11.11.11.2 is-at 1e:12:5b:e9:80:ca (oui Unknown), length 28
13:24:34.9780 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 31132, seq 1, length 64
13:24:34.9789 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 31132, seq 1, length 64
```

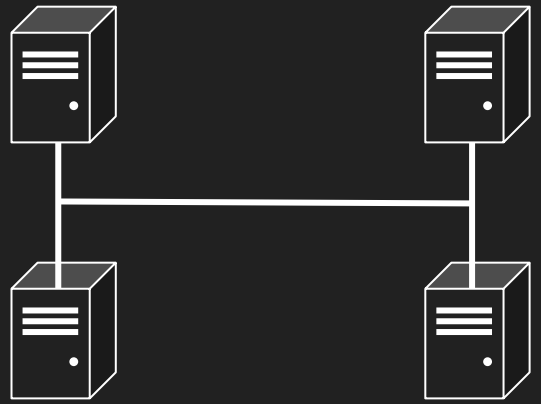


Bridge

여러대를 연결하려면 어떡할까요? Hub

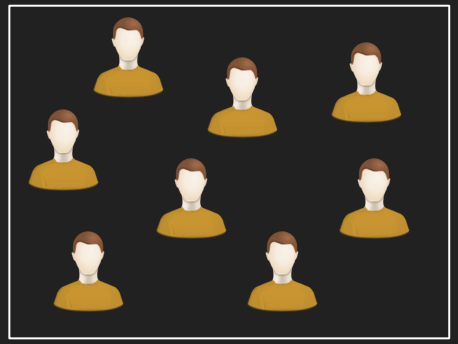
Hub : 여러대 PC를 연결





Bridge

Hub : 여러대 PC를 연결



연결할 장비가 계속 늘어나면?

Bridge

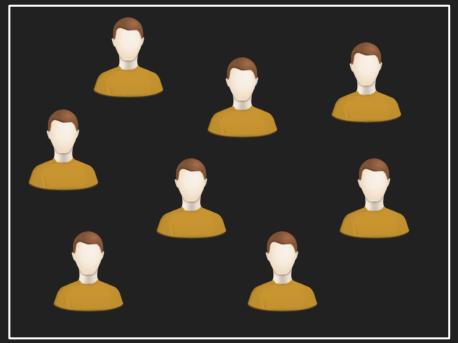
Hub : 여러대 PC를 연결

연결할 장비가 계속 늘어나면

Why? 이더넷의 특징 (CSMA/CD, 눈치게임)

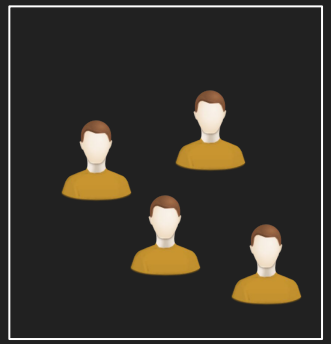
Bridge

Hub : 여러대 PC를 연결



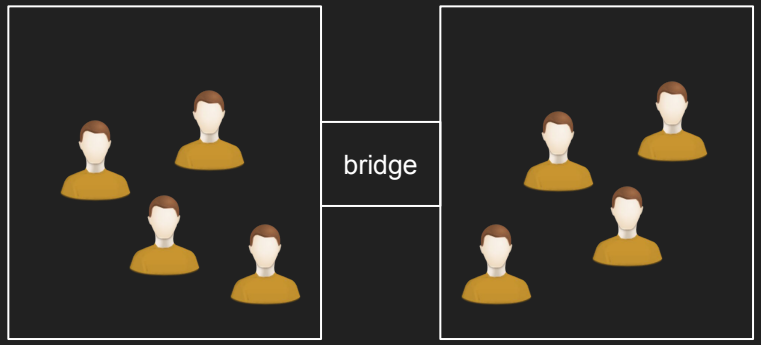
연결할 장비가 계속 늘어나면?

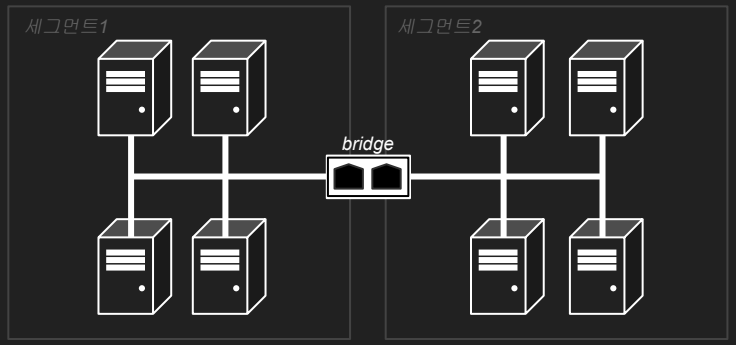
방을 나눠주면 좋겠다



Bridge

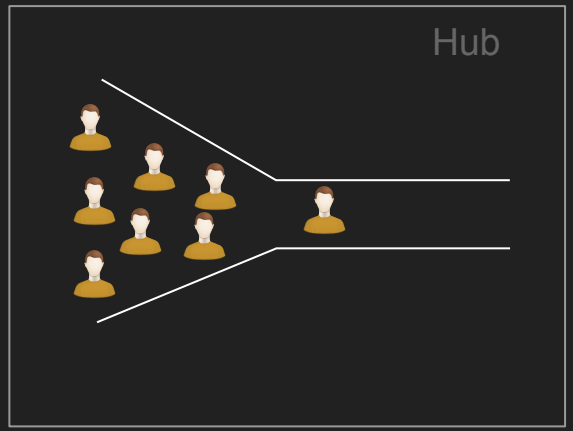
Bridge : Collision Domain을 쪼개줌

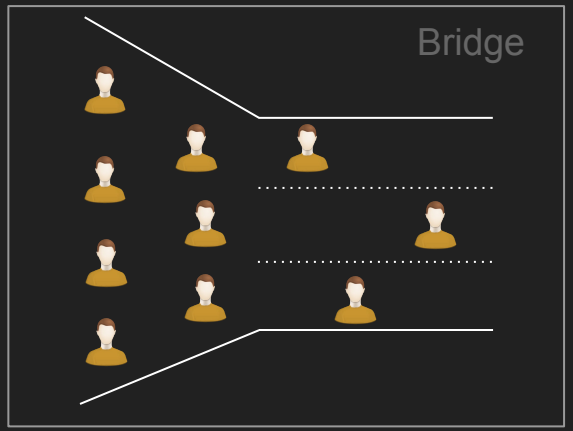




Bridge

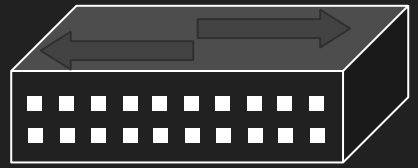
Bridge : Collision Domain을 쪼개줌





Bridge

Bridge : OSI L2계층 (Data Link) 을 처리



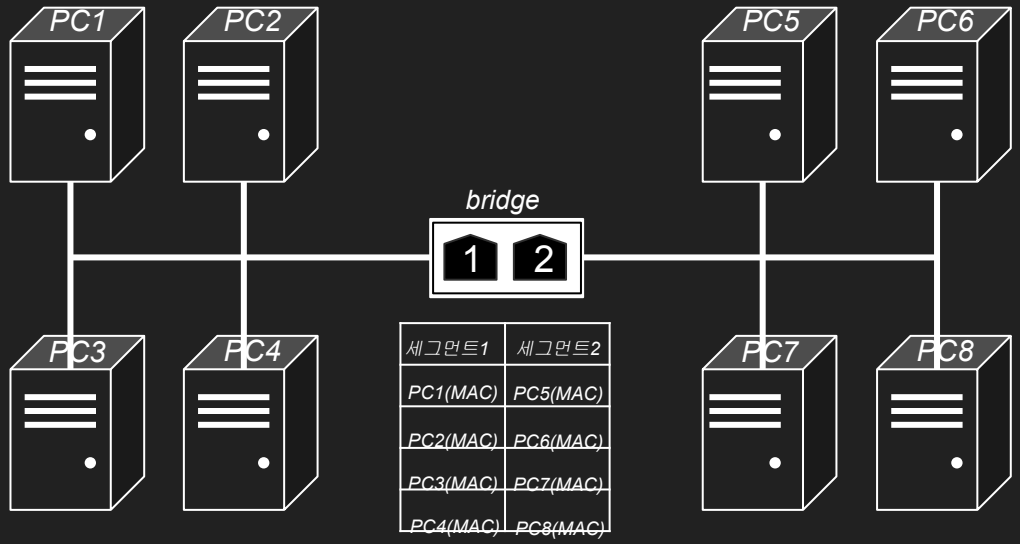
Frame 구조

Preamble	SFD	dest. MAC	src. MAC	Type Length	Data / Payload	CRC

패킷 유입 > 목적지 MAC 주소 확인 > 주소(MAC) 테이블과 비교 > 해당 디바이스가 연결된 포트로 패킷 전달

Bridge

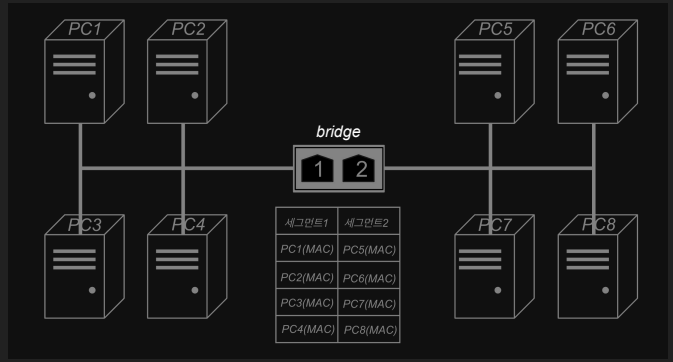
서버는 포트에 연결되고 포트별로 어떤 세그먼트에 속하는지 관리



Bridge

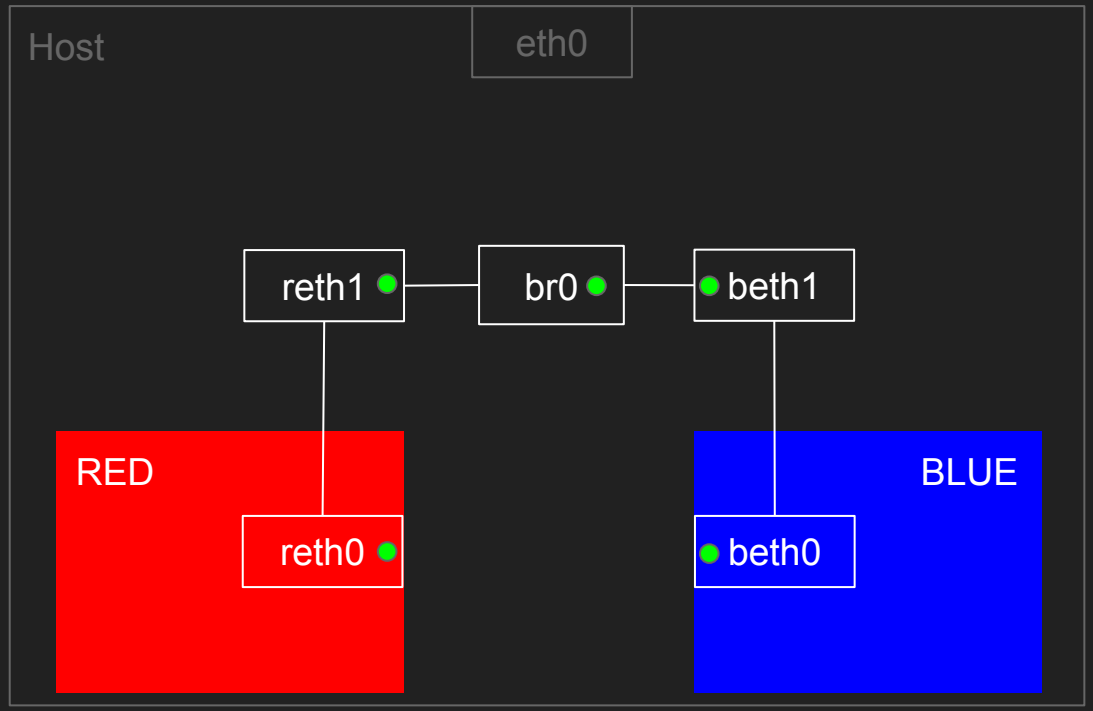
주요기능

- Learning : 각 MAC이 어떤 포트, 세그먼트에 있는지 학습
- Flooding : 목적지(MAC)을 알아내기 위한 Broadcast (발신 포트제외)
- Forwarding : 목적지 포트로 전송
- Filtering : 동일 세그먼트의 목적지인 경우 세그먼트 내에서 처리
- Aging : TTL (300s) 내에 통신이 없으면 delete



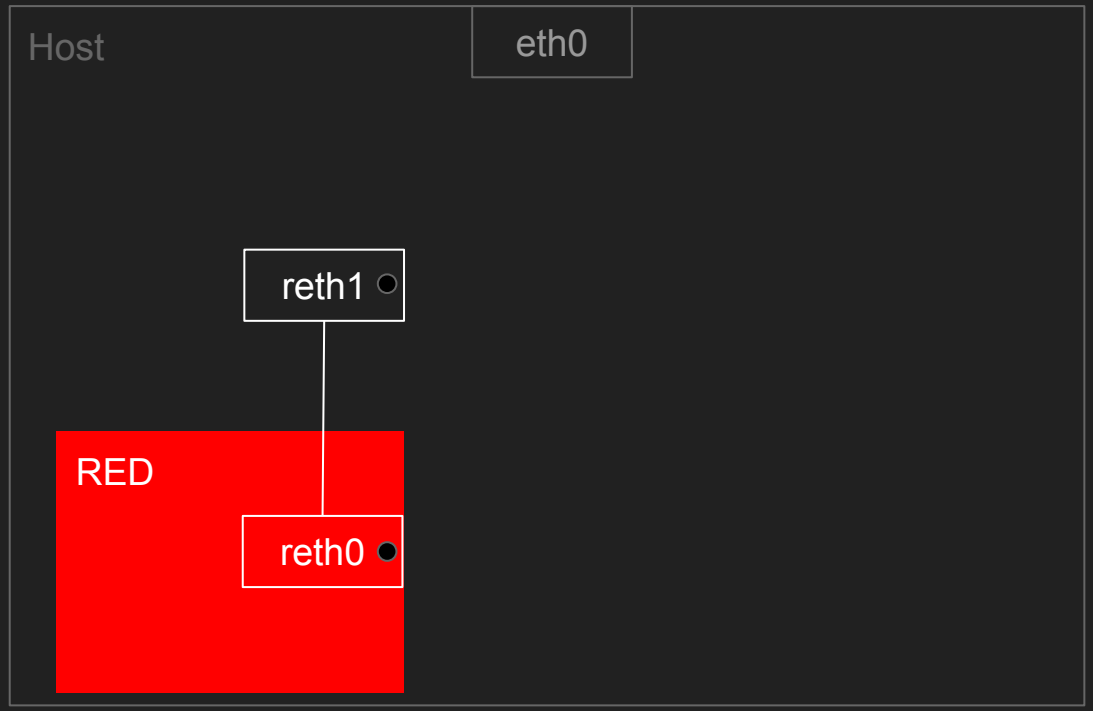
(실습2) Bridge 통신

Bridge를 경유하여 RED/BLUE 간 통신을 해보아요



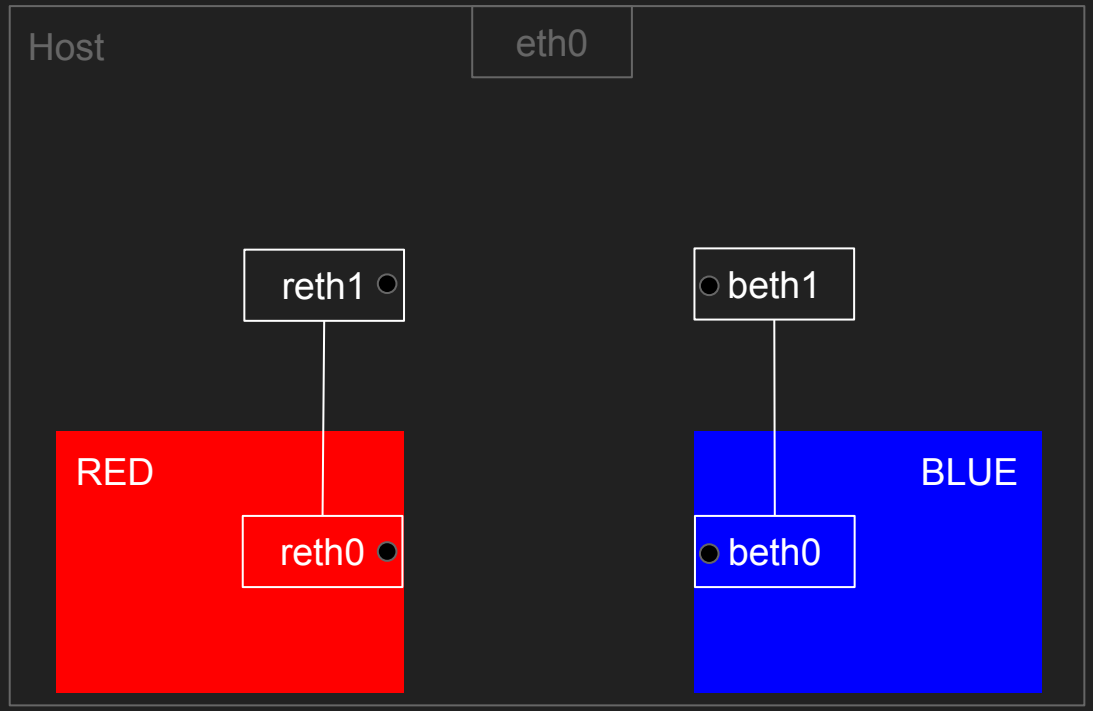
(실습2) Bridge 통신

```
# ip netns add RED
# ip link add reth0 type veth peer name reth1
# ip link set reth0 netns RED
```



(실습2) Bridge 통신

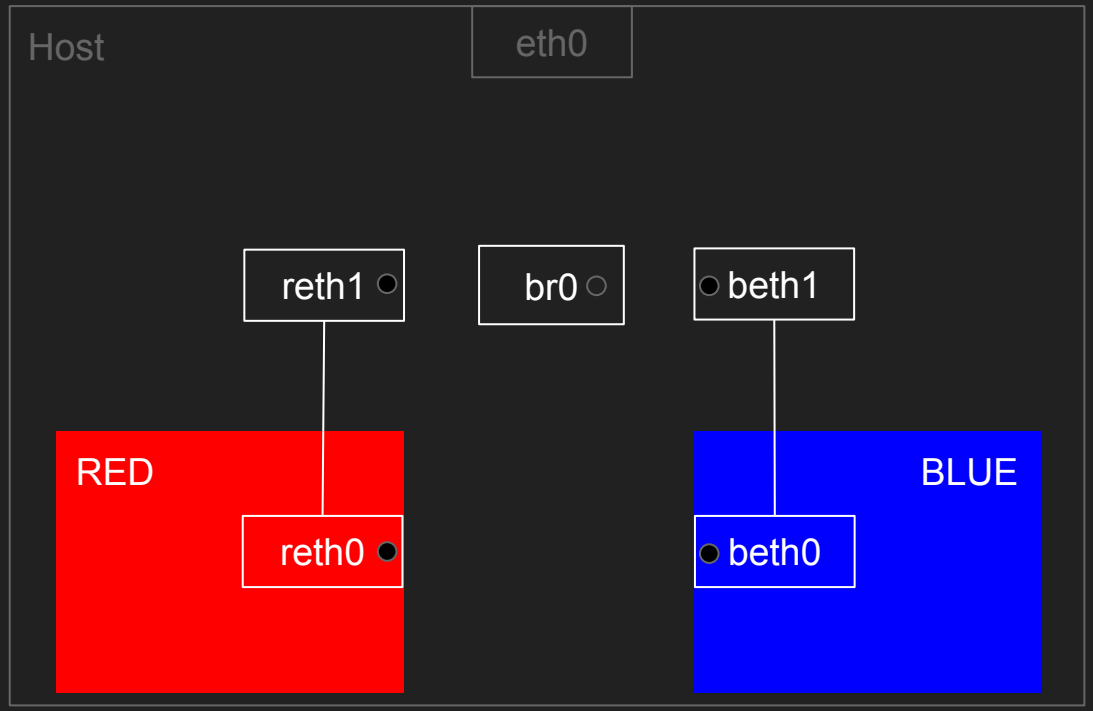
```
# ip netns add BLUE
# ip link add beth0 type veth peer name beth1
# ip link set beth0 netns BLUE
```



(실습2) Bridge 통신

```
# ip link add br0 type bridge
```

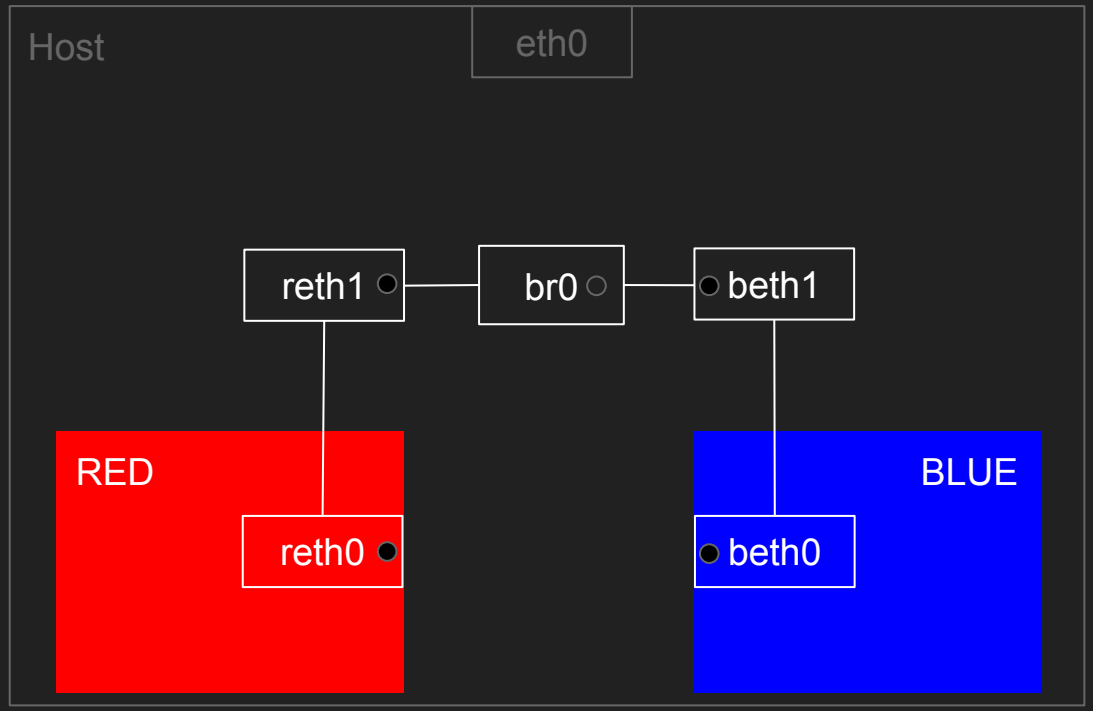
add Bridge



(실습2) Bridge 통신

```
# ip link set reth1 master br0
# ip link set beth1 master br0
```

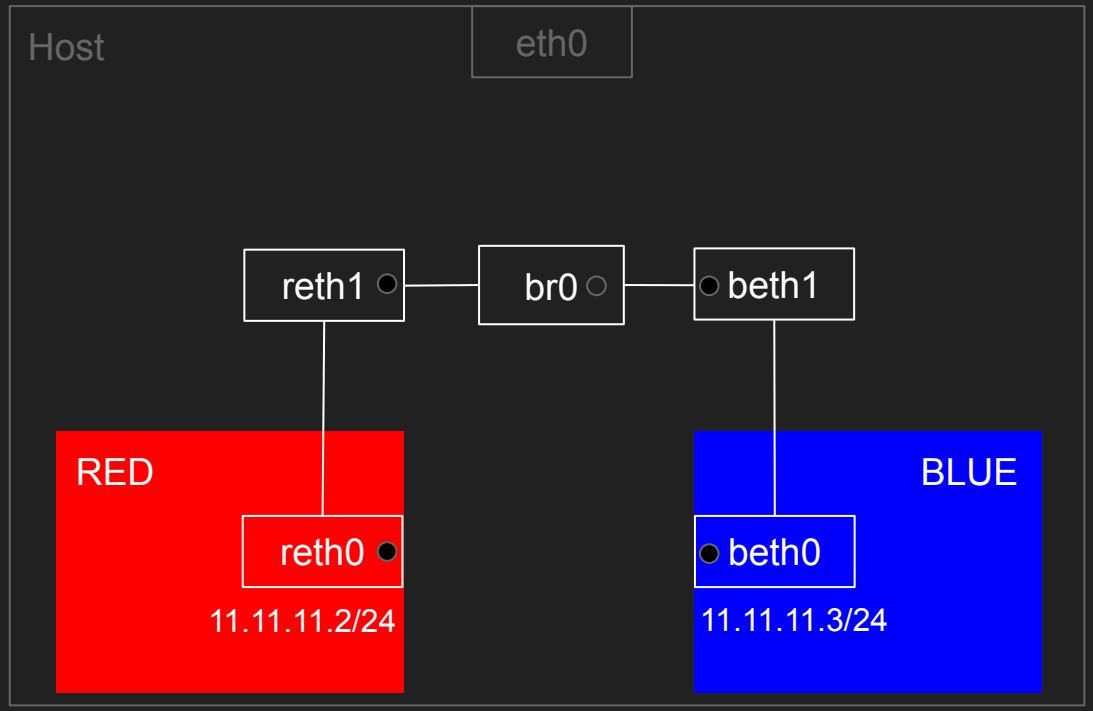
Attach to the bridge



(실습2) Bridge 통신

```
# ip netns exec RED ip addr add 11.11.11.2/24 dev reth0
# ip netns exec BLUE ip addr add 11.11.11.3/24 dev beth0
```

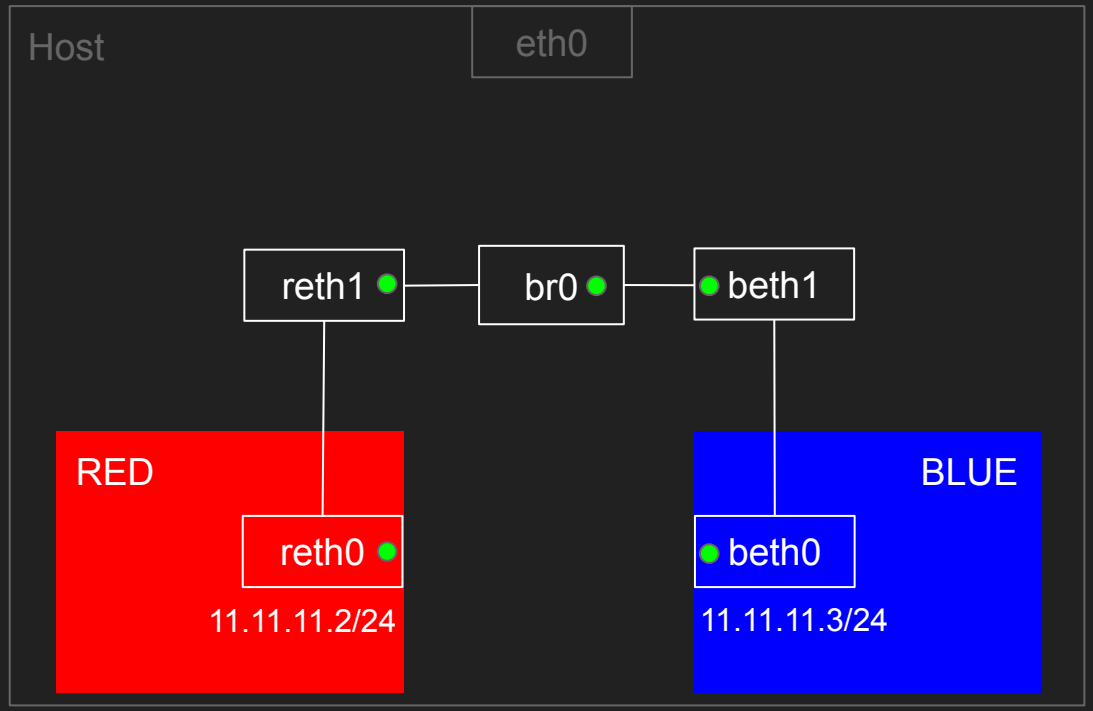
Set IP Address



(실습2) Bridge 통신

```
# ip netns exec RED ip link set reth0 up; ip link set reth1 up ;
# ip netns exec BLUE ip link set beth0 up; ip link set beth1 up ;
# ip link set br0 up
```

Turn on !



(실습2) Bridge 통신

터미널 창을 3개를 준비하고 실습환경(VM)으로 접속해주세요

터미널 #1

```
#
```

터미널 #2

```
#
```

터미널 #2

```
#
```

(실습2) Bridge 통신

*자 … **RED, BLUE**에 입장하여 봅시다*

터미널 #1 (RED, 11.11.11.2)

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

터미널 #2 (HOST)

```
#
```

터미널 #2 (BLUE, 11.11.11.3)

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

(실습2) Bridge 통신

ip address show ~ 각 네임스페이스의 네트워크 장치를 확인합니다

터미널 #1 (RED, 11.11.11.2)

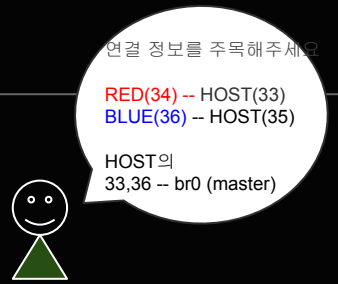
```
# ip address show
...
34: reth0@if33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether d2:e5:f4:1c:d6:93 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 11.11.11.2/24 scope global reth0
```

터미널 #2 (BLUE, 11.11.11.3)

```
# ip address show
...
36: beth0@if35: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 3a:c6:51:d6:08:40 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 11.11.11.3/24 scope global beth0
```

터미널 #2 (HOST)

```
# ip address show

33: reth1@if34: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000
    link/ether 8a:65:75:f5:a2:e2 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::8865:75ff:fef5:a2e2/64 scope link
       valid_lft forever preferred_lft forever
35: beth1@if36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000
    link/ether 0a:b6:9f:56:7d:85 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::8b6:9fff:fe56:7d85/64 scope link
       valid_lft forever preferred_lft forever
37: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 0a:b6:9f:56:7d:85 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8b6:9fff:fe56:7d85/64 scope link
       valid_lft forever preferred_lft forever
```



(실습2) Bridge 통신

ip neighbour show ~ "네 이웃을 ~~사랑~~확인하라"

터미널 #1 (RED, 11.11.11.2)

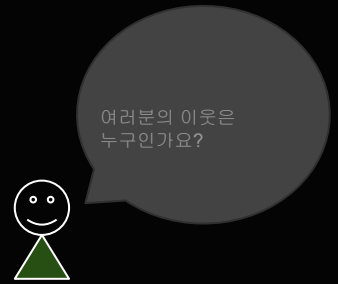
```
# ip neighbour show
```

터미널 #2 (HOST)

```
# ip neighbour show
```

터미널 #3 (BLUE, 11.11.11.3)

```
# ip neighbour show
```



ip-neighbour neighbour/arp tables management *참고) # man ip-neighbour*

- 줄여서 "ip neigh" 로 많이 씀
- to ADDRESS : 이웃의 IP 주소입니다
- dev NAME : 이웃과 연결해주는 장치(interface) 이름입니다
- lladdr LLADDRESS : ll(엘엘) ~ Link Layer (2계층) 이웃의 MAC주소입니다. null 값일 수도 있습니다
- nud STALE : nud (Neighbour Unreachability Detection "왜 이렇게 이웃을 감시할까요ㅋ") 이웃의 부재상태를 기록합니다

(실습2) Bridge 통신

ip route show ~ 각 네임스페이스의 라우팅 테이블을 확인합니다

터미널 #1 (RED, 11.11.11.2)

```
# ip route show

11.11.11.0/24 dev reth0 proto kernel scope link src 11.11.11.2
```

터미널 #2 (BLUE, 11.11.11.3)

```
# ip route show

11.11.11.0/24 dev beth0 proto kernel scope link src 11.11.11.3
```

터미널 #2 (HOST)

```
# ip route show

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
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
```

ip-route /etc/iproute2/rt_* *참고) # man ip-route*

- dev : output device NAME
- proto : the route was installed by (or due to) … ex) kernel, redirect (icmp), boot, dhcp, static …
- scope : valid in (global - everywhere, link - device , host, site ...)
- via : ADDRESS. nexthop (router)
- src : ADDRESS. source

(실습2) Bridge 통신

터미널 #1 (RED, 11.11.11.2)

```
# bridge fdb show
```

터미널 #2 (HOST)

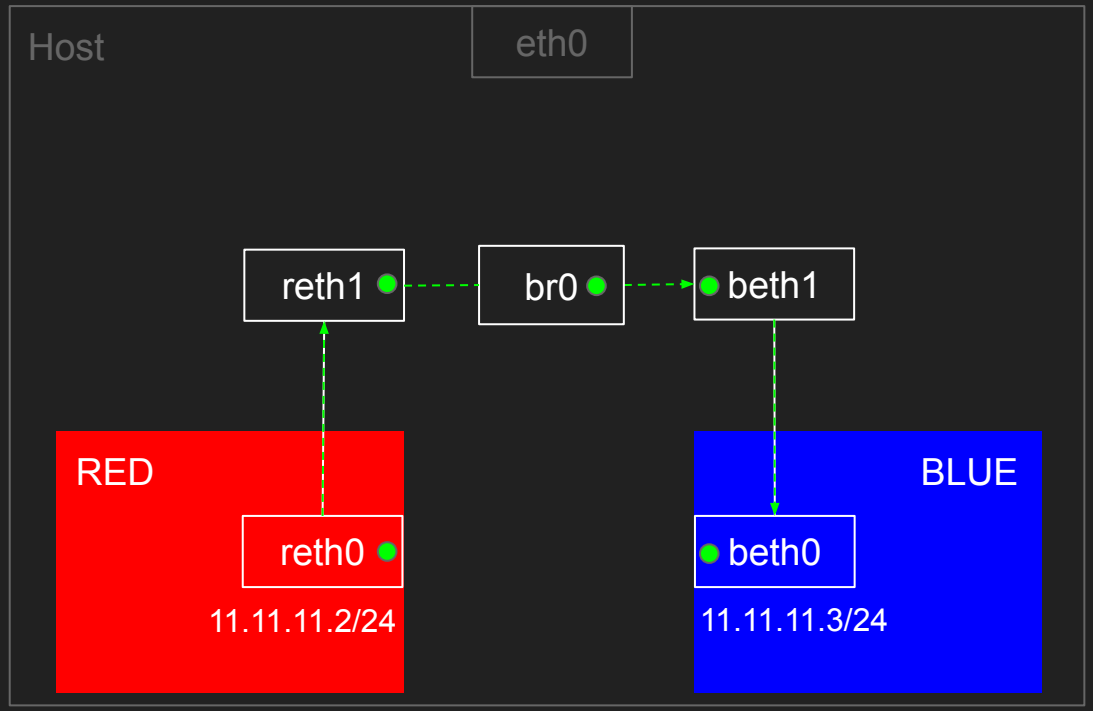
```
# bridge fdb show
```

터미널 #3 (BLUE, 11.11.11.3)

```
# bridge fdb show
```

(실습2) Bridge 통신

Ping test



(실습2) Bridge 통신

RED → BLUE ping failed why?

터미널 #1 (RED, 11.11.11.2)

```
# ping -c 1 11.11.11.3

PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data


--- 11.11.11.3 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

터미널 #2 (호스트)

```
# tcpdump -l -i br0

12:31:17.000959 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
12:31:17.001016 ARP, Reply 11.11.11.3 is-at ea:86:a4:a9:08:b8 (oui Unknown), length 28
12:31:17.001021 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 1289, seq 1, length 64
```

터미널 #3 (BLUE, 11.11.11.3)

```
# tcpdump -l -i beth0

12:31:17.000966 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
12:31:17.001014 ARP, Reply 11.11.11.3 is-at ea:86:a4:a9:08:b8 (oui Unknown), length 28
```

ARP 만 통과

(실습2) Bridge 통신

RED → BLUE ping failed why?

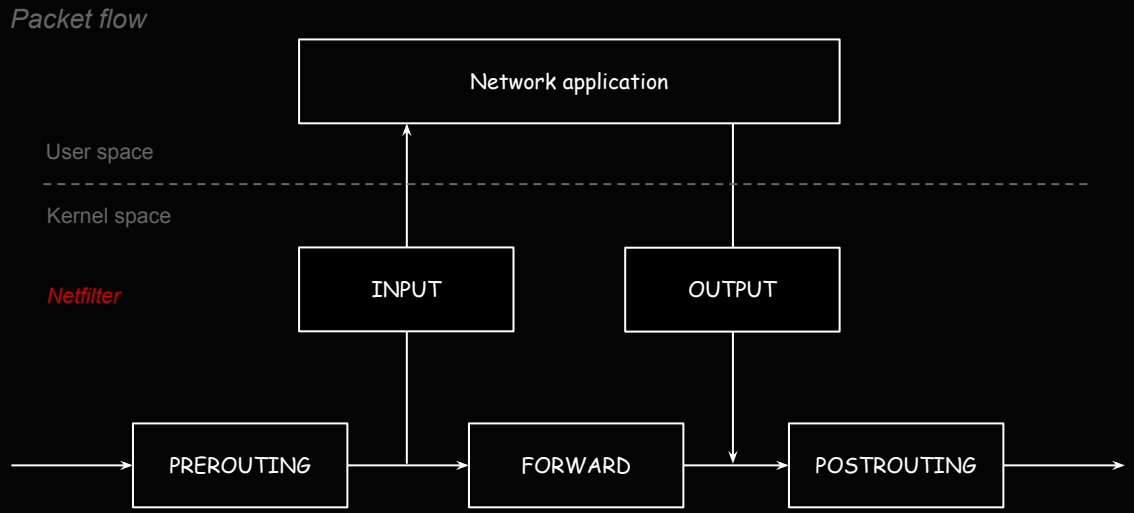
ICMP (Internet Control Message Protocol)

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

(실습2) Bridge 통신

RED → BLUE ping failed why?



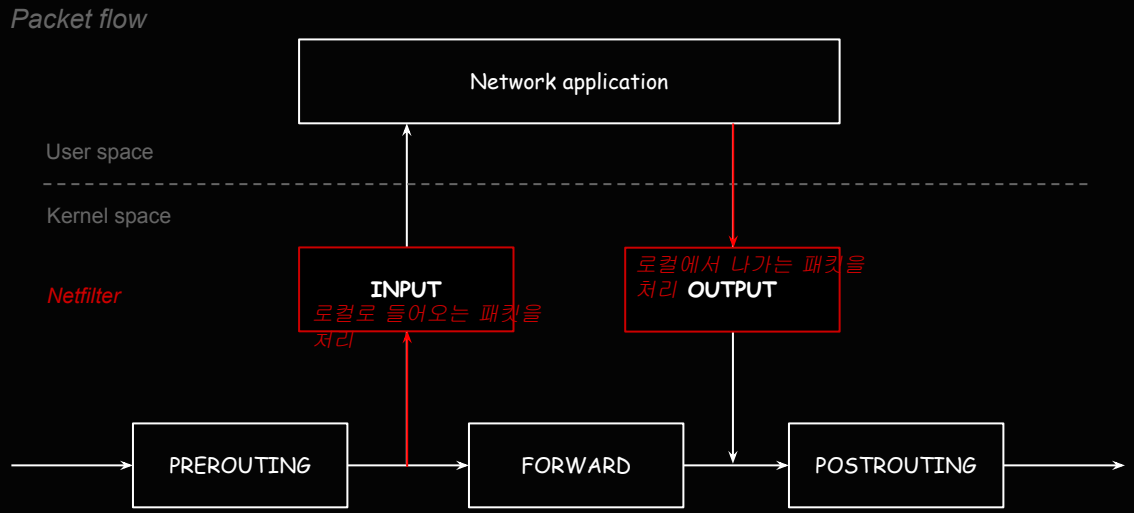
Netfilter

커널 모듈로 네트워크 패킷을 처리하는 프레임워크.
네트워크 연산을 핸들러 형태로 처리할 수 있도록 **hook 지점**을 제공
패킷이 어떻게 전송될 지에 대한 결정방법을 제공

참고 : https://saysecurity.tistory.com/15

(실습2) Bridge 통신

RED → BLUE ping failed why?



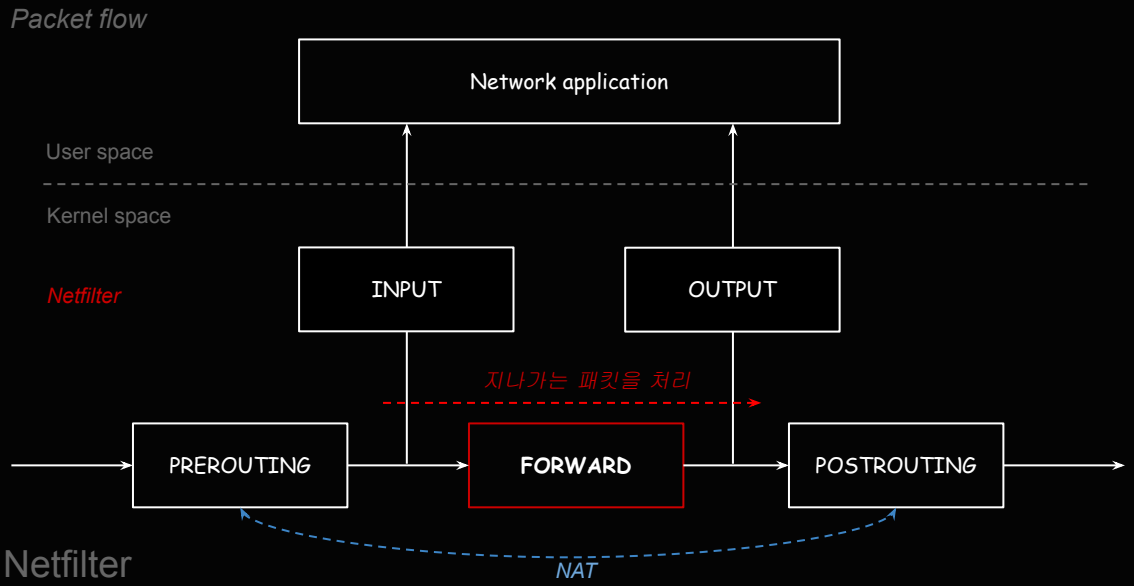
Netfilter

커널 모듈로 네트워크 패킷을 처리하는 프레임워크.
네트워크 연산을 핸들러 형태로 처리할 수 있도록 **hook 지점**을 제공
패킷이 어떻게 전송될 지에 대한 결정방법을 제공

참고 : https://saysecurity.tistory.com/15

(실습2) Bridge 통신

RED → BLUE ping failed why?



Netfilter

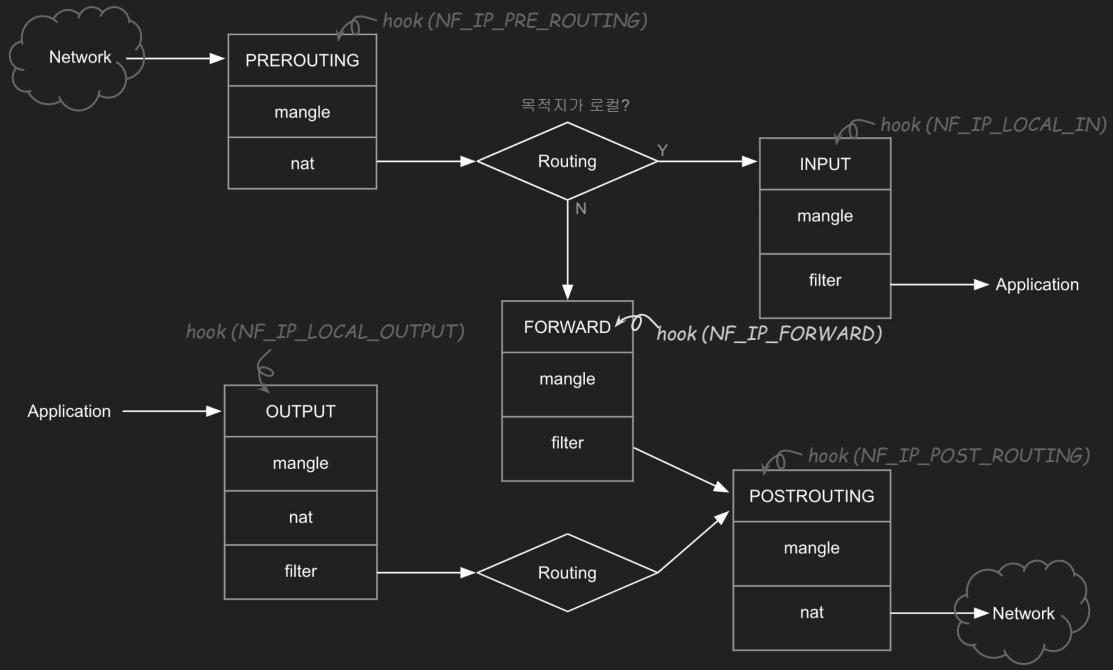
커널 모듈로 네트워크 패킷을 처리하는 프레임워크.
네트워크 연산을 핸들러 형태로 처리할 수 있도록 **hook 지점**을 제공
패킷이 어떻게 전송될 지에 대한 결정방법을 제공

참고 : https://saysecurity.tistory.com/15

(실습2) Bridge 통신

RED → BLUE ping failed why?

패킷은 netfilter에서 파놓은 여러 *hook* 을 통과하는데 *hook* 별로 iptables에서 정의한 각 체인룰을 점검합니다



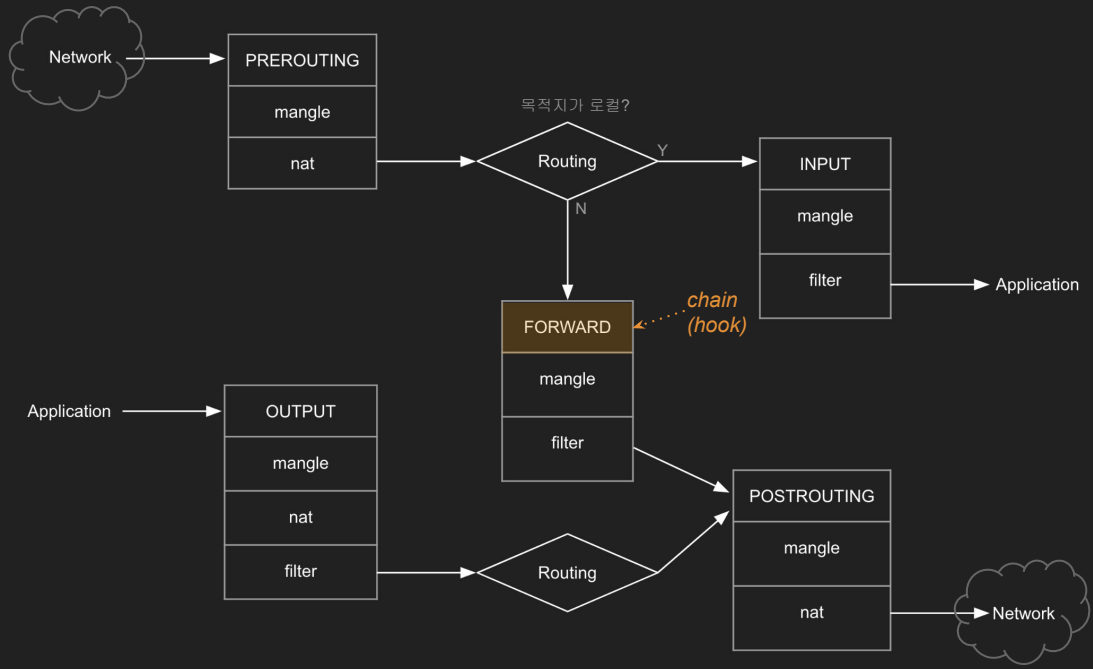
iptables ?

Netfilter가 파놓은 "hook"에 룰을 등록하고 관리하는 방법을 제공하는 툴

(실습2) Bridge 통신

RED → BLUE ping failed why?

hook 에 정의된 체인은 테이블 순서대로 등록된 룰을 체크하고 조건을 만족하면 action(target)을 트리거합니다



** table : 목적/용도별 rules 모음* 예) filter, nat, mangle, . . .

** chain : 패킷이 지나가는 hook별로 존재* 예) PREROUTING, FORWARD, INPUT, … (넷필터의 hook에 매핑됩니다)

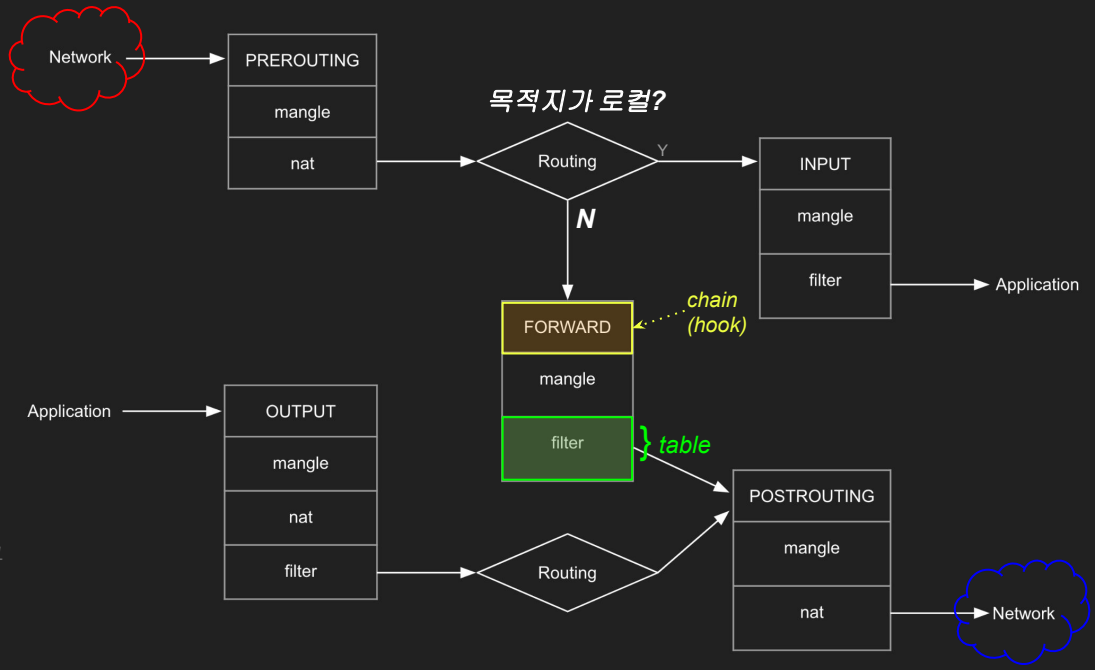
** rule : table 과 chain matrix에 대해서 정의함* 예) protocol type, dest./src. address, headers …

** action (target) : 패킷이 룰에 매칭되면 트리거됨* 예) ACCEPT, DROP, REJECT, … * -j : jump

(실습2) Bridge 통신

RED → BLUE ping failed why?

"외부(src) → 외부(dst)" 패킷이므로
FORWARD 체인의 filter 테이블 룰을
봐야 합니다



RED → BLUE 통신은 Bridge를 경유하는데
Bridge가 호스트 네임스페이스에 있기때문에
패킷은 외부(RED 네임스페이스)에서 왔고
목적지 또한 외부(BLUE 네임스페이스)이므로
(로컬)호스트를 경유, 즉 FORWARD로 처리됩니다

FOWARD : NF_IP_FORWARD (hook)에 등록된 체인
~ NF_IP_FORWARD : incoming 패킷이 다른 호스트로 포워딩되는 경우에 트리거되는 netfilter hook
filter (table) : 패킷을 목적지로 전송 여부를 결정

filter 테이블에 FORWARD 정책 확인

터미널 #2 (호스트)

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

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

filter 테이블의 FORWARD 체인의 기본 정책(policy)이 DROP 이네요

(실습2) Bridge 통신

filter 테이블에 FORWARD 룰 등록

터미널 #2 (호스트)

```
# iptables -t filter -A FORWARD -s 11.11.11.0/24 -j ACCEPT
```

table / APPEND chain rule / src. address / jump to ACCEPT (허용)


```
# iptables -t filter -L

. . .
Chain FORWARD (policy DROP)
target     prot opt source               destination
. . .
ACCEPT     all  --  11.11.11.0/24       anywhere
```

(실습2) Bridge 통신

RED → BLUE ~ OK

터미널 #1 (RED, 11.11.11.2)

```
# ping -c 1 11.11.11.3

PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data
64 bytes from 11.11.11.3: icmp_seq=1 ttl=64 time=0.189 ms
```

터미널 #2 (호스트)

```
# tcpdump -l -i br0

12:48:10.349976 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
12:48:10.350007 ARP, Reply 11.11.11.3 is-at 3a:c6:51:d6:08:40 (oui Unknown), length 28
12:48:10.350012 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 31892, seq 1, length 64
12:48:10.350033 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 31892, seq 1, length 64
12:48:15.376006 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
12:48:15.376089 ARP, Reply 11.11.11.2 is-at d2:e5:f4:1c:d6:93 (oui Unknown), length 28
```

터미널 #3 (BLUE, 11.11.11.3)

```
# tcpdump -l -i beth0

12:48:10.349983 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
12:48:10.350006 ARP, Reply 11.11.11.3 is-at 3a:c6:51:d6:08:40 (oui Unknown), length 28
12:48:10.350025 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 31892, seq 1, length 64
12:48:10.350032 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 31892, seq 1, length 64
12:48:15.375989 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
12:48:15.376092 ARP, Reply 11.11.11.2 is-at d2:e5:f4:1c:d6:93 (oui Unknown), length 28
```

(실습2) Bridge 통신

```
# ping -c 1 11.11.11.3
```


```
[R] 13:04:37.520104 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
[~] 13:04:37.520111 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
[B] 13:04:37.520118 ARP, Request who-has 11.11.11.3 tell 11.11.11.2, length 28
```

#1 ARP request

```
[B] 13:04:37.520134 ARP, Reply 11.11.11.3 is-at 96:04:cf:2b:35:8a (oui Unknown), length 28
[~] 13:04:37.520135 ARP, Reply 11.11.11.3 is-at 96:04:cf:2b:35:8a (oui Unknown), length 28
[R] 13:04:37.520136 ARP, Reply 11.11.11.3 is-at 96:04:cf:2b:35:8a (oui Unknown), length 28
```

#2 ARP response

```
[R] 13:04:37.520140 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 32264, seq 1, length 64
[~] 13:04:37.520140 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 32264, seq 1, length 64
[B] 13:04:37.520157 IP 11.11.11.2 > 11.11.11.3: ICMP echo request, id 32264, seq 1, length 64
```

#3 ICMP request

```
[B] 13:04:37.520165 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 32264, seq 1, length 64
[~] 13:04:37.520166 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 32264, seq 1, length 64
[R] 13:04:37.520168 IP 11.11.11.3 > 11.11.11.2: ICMP echo reply, id 32264, seq 1, length 64
```

#4 ICMP response

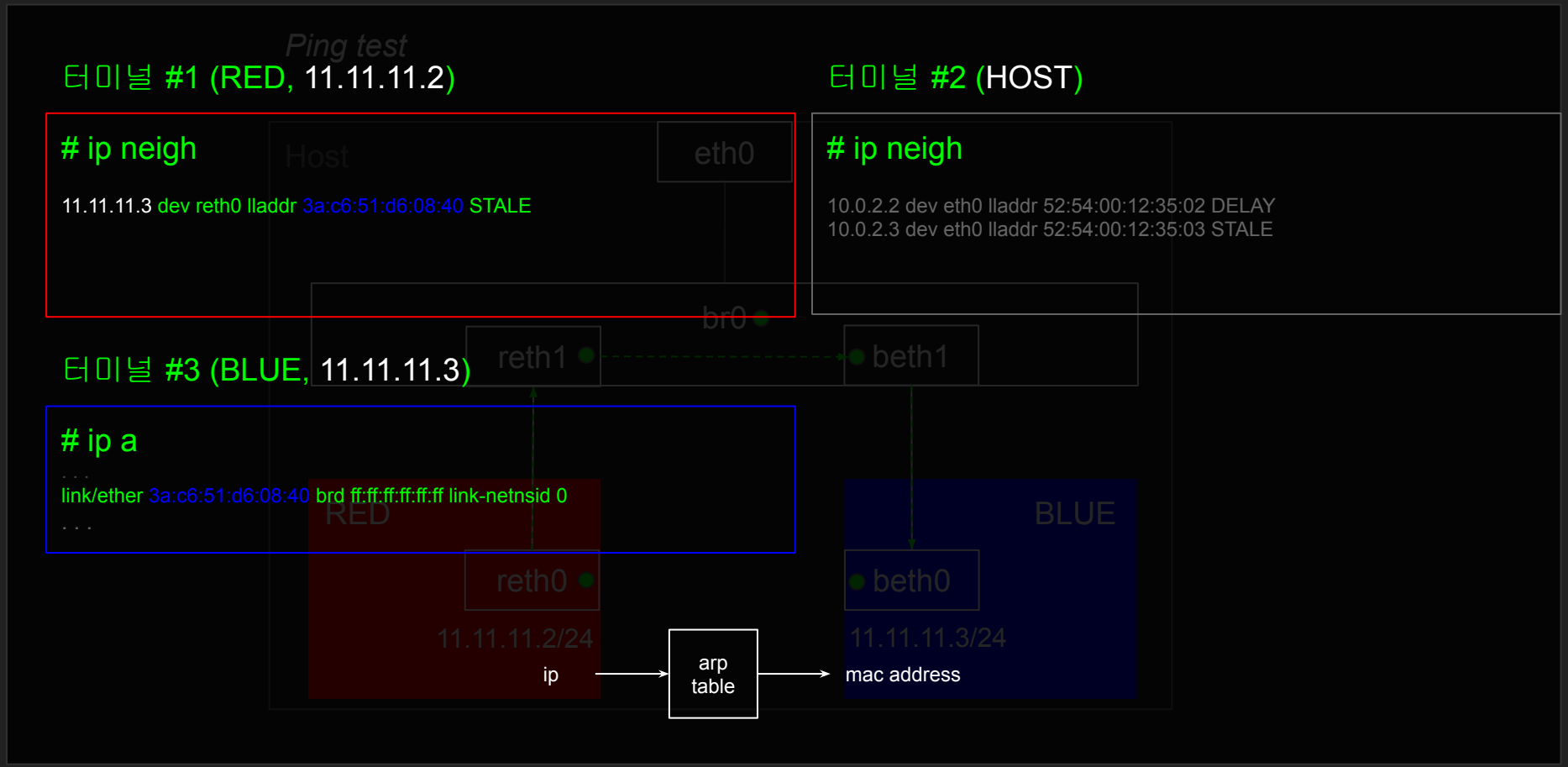
```
[B] 13:04:42.767853 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
[~] 13:04:42.767876 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
[R] 13:04:42.767935 ARP, Request who-has 11.11.11.2 tell 11.11.11.3, length 28
```

#5 ARP request

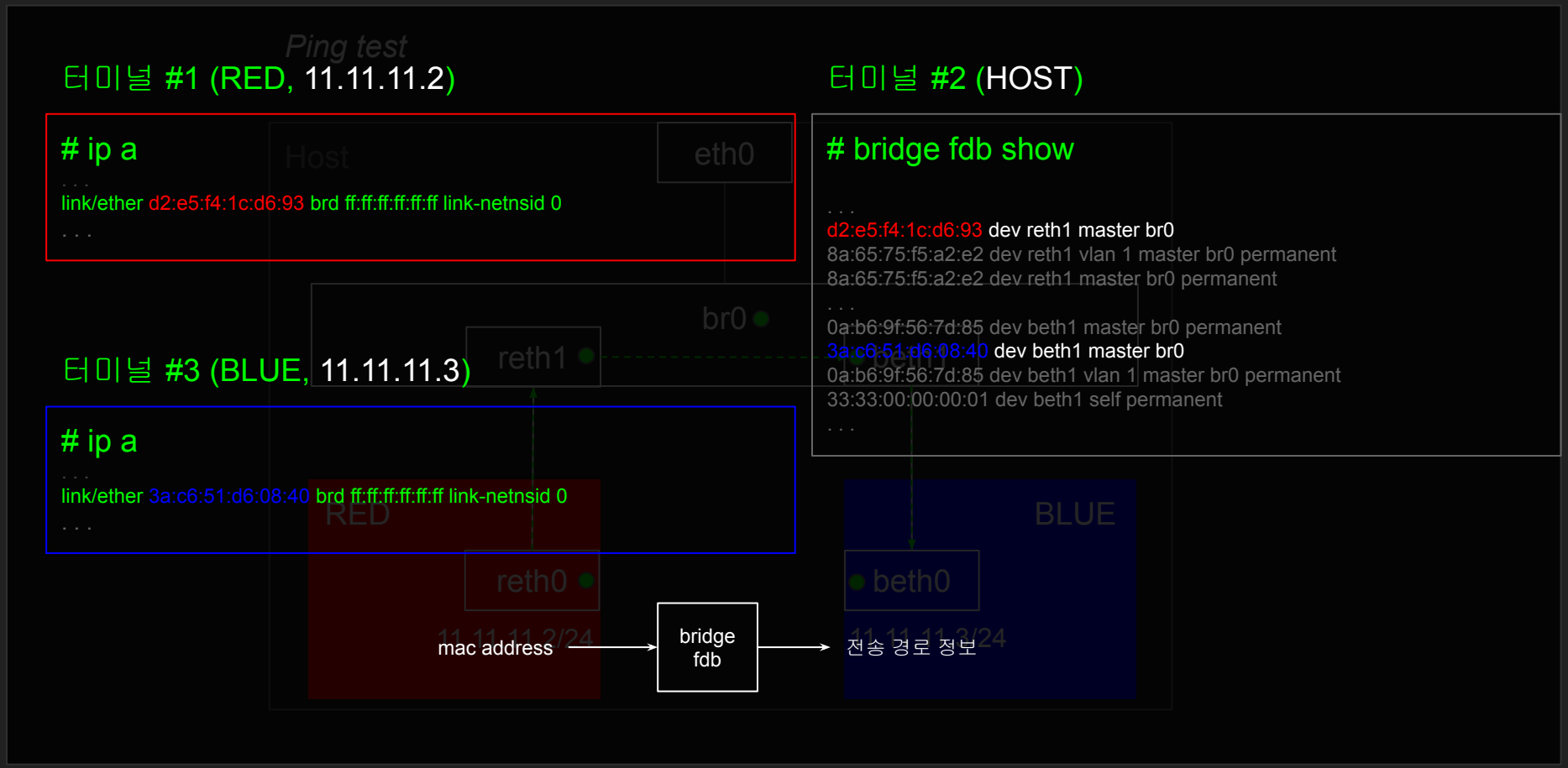
```
[R] 13:04:42.767958 ARP, Reply 11.11.11.2 is-at 2e:4c:03:60:79:a8 (oui Unknown), length 28
[~] 13:04:42.767960 ARP, Reply 11.11.11.2 is-at 2e:4c:03:60:79:a8 (oui Unknown), length 28
[B] 13:04:42.767962 ARP, Reply 11.11.11.2 is-at 2e:4c:03:60:79:a8 (oui Unknown), length 28
```

#6 ARP response

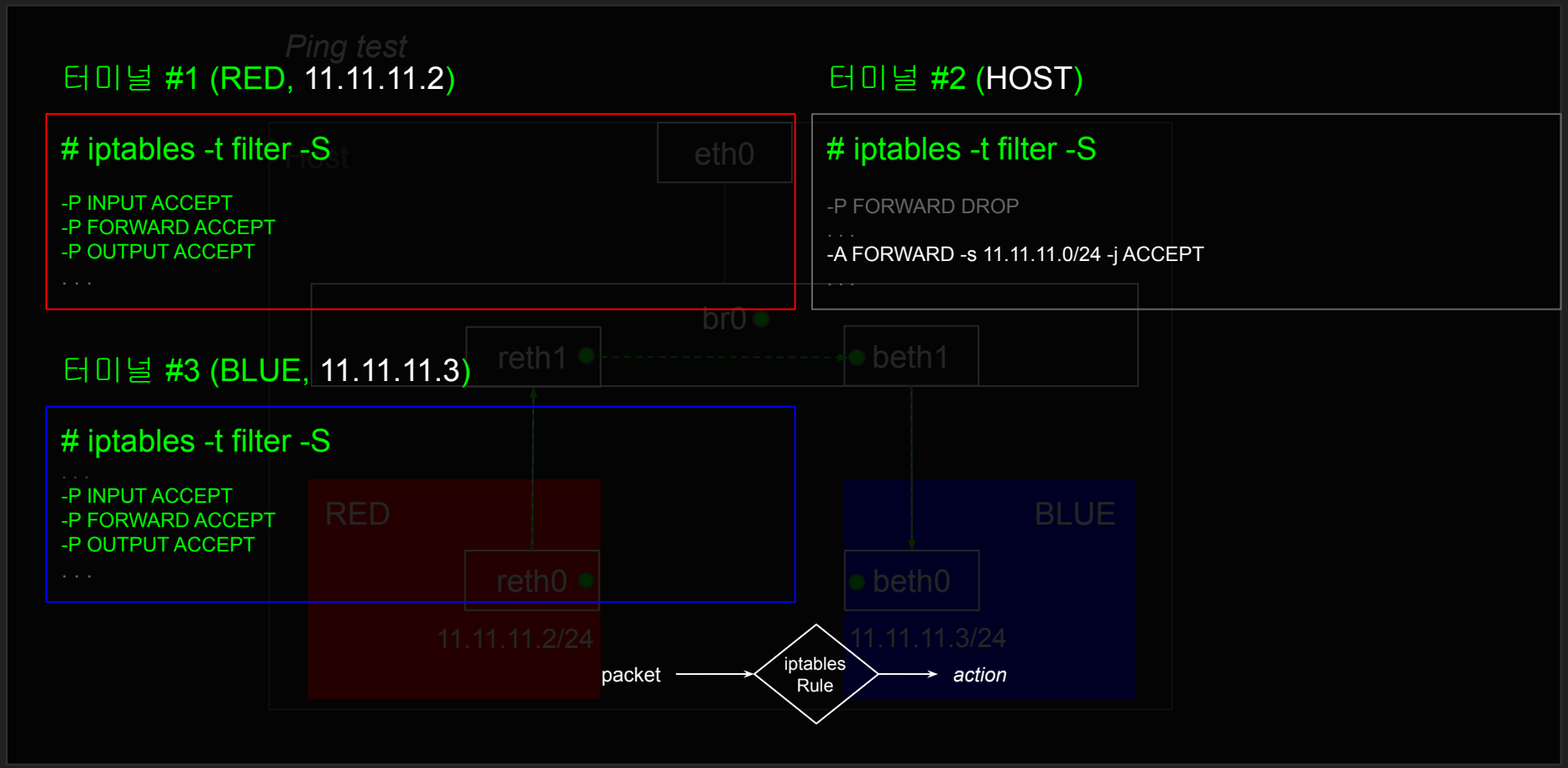
(실습2) Bridge 통신



(실습2) Bridge 통신

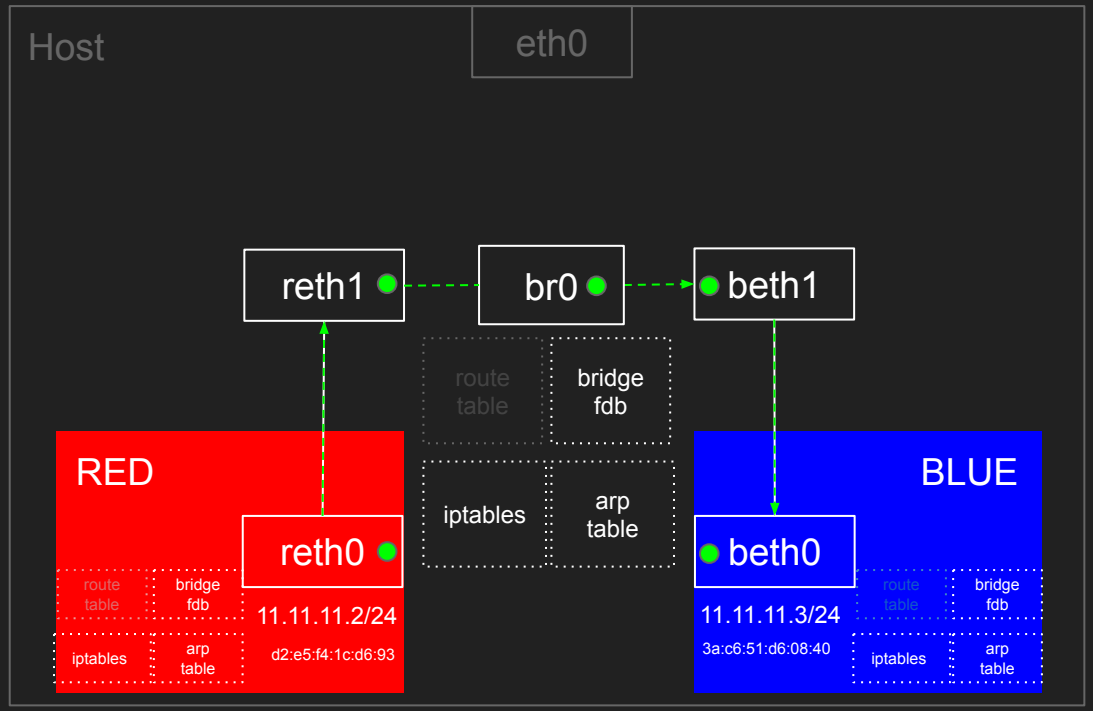


(실습2) Bridge 통신



(실습2) Bridge 통신 ~ (성공)

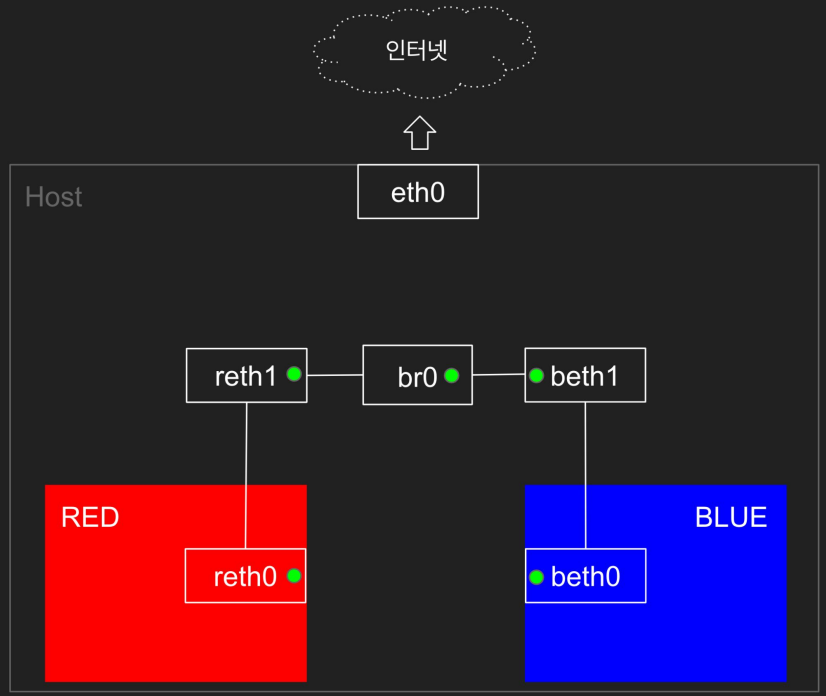
arp, fdb정보를 이용하여 통신하고 iptables rule에 따라 전송여부 결정



4편 예고

지금까지 호스트 안의 또다른 가상 네트워크를 만들어 보았는데요

4편에서는 가상 네트워크와 외부 네트워크의 통신을 다뤄보도록 하겠습니다



수고 많으셨습니다 다음편에서 봬요 :-)

Thanks