experiment environment：

```
host1:
docker@test1:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:00:AD:E0
          inet addr:192.168.99.111  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe00:ade0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3238 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3248 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:478487 (467.2 KiB)  TX bytes:476469 (465.3 KiB)

docker@test1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
ATUS              PORTS               NAMES
4e88a6be6b14        busybox             "sleep 60000000"    36 minutes ago      Up
 36 minutes                            c1
docker@test1:~$ docker exec c1 ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

veth2     Link encap:Ethernet  HWaddr 92:1E:A1:74:BB:4D
          inet addr:10.0.10.10  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4093 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4467 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:398746 (389.4 KiB)  TX bytes:435750 (425.5 KiB)

docker@test1:~$


host2:
docker@test2:~$ ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 08:00:27:23:B0:8F
          inet addr:192.168.99.110  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe23:b08f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3336 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3346 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:492815 (481.2 KiB)  TX bytes:490797 (479.2 KiB)

docker@test2:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
ATUS              PORTS               NAMES
515610caa297        busybox             "sleep 60000000"    36 minutes ago      Up
 36 minutes                            c2
docker@test2:~$ docker exec c2 ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

veth2     Link encap:Ethernet  HWaddr 0A:42:35:1E:C2:E4
          inet addr:10.0.10.11  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4021 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4005 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:392082 (382.8 KiB)  TX bytes:390866 (381.7 KiB)

docker@test2:~$
```

> host1: docker run -d --net none --name c1 busybox sleep 60000000
>
> host2: docker run -d --net none --name c2 busybox sleep 60000000

start two containers with no network.

> ip link add dev veth1 mtu 1500 type veth peer name veth2 mtu 1500

use ip link to create a veth pair device (veth1, veth2)

> pid=$(docker inspect c1_cid --format '{{.State.Pid}}')

get the process id for the container(c1_cid)

> ln -sf /proc/${pid}/ns/net /var/run/netns/tmp-ns

ln the network namespace of the container to /var/run/netns/tmp-ns

> ip link set dev veth2 netns tmp-ns

set veth2 to the network space of the container, after this operation, veth1 is in the host namespace, and

veth2 is in the container namespace.

```
docker@test1:~$ docker exec c1 ifconfig -a
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

sit0      Link encap:IPv6-in-IPv4
          NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

veth2     Link encap:Ethernet  HWaddr 0E:69:D7:84:35:F0
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

> ip netns exec tmp-ns ip addr add dev veth2 10.0.10.10/24 ip netns exec tmp-ns ip link up veth2

should be "ip netns exec tmp-ns ip link set veth2 up"

set veth2 with the given IP and bring up the interface.

with the above steps, a veth pair is created, with one end(veth2) in the container and the other end(veth1) in the host.

> ip netns add my-overlay

add a new network namespace "my-overlay"

> ip netns exec my-overlay ip link add dev br0 type bridge

create a brdige device br0

```
docker@test1:~$ sudo ip netns exec my-overlay ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT group default ql
en 1000
    link/sit 0.0.0.0 brd 0.0.0.0
3: br0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group de
fault qlen 1000
    link/ether ca:7c:47:49:51:8a brd ff:ff:ff:ff:ff:ff
```

> ip netns exec my-overlay ip addr add dev br0 10.0.10.1/24

set a ip address for bridge br0

> ip netns exec my-overlay ip link set br0 up

bring up the bridge device br0

> ip link add dev vxlan1 type vxlan id 42 proxy learning dstport 4789
>
> ip link set vxlan1 netns my-overlay

create a vxlan device to use id 42 and tunnel traffic on port 4789.

proxy option allows the vxlan interface to answer arp queries.

learning option allows the update of forward database(fdb) of vxlan interface.

move vxlan interface to namespace "my-overlay"

> ip netns exec my-overlay ip link set vxlan1 master br0

set vxlan1 as the br0's slave

> ip netns exec my-overlay ip link set vxlan1 up

bring up vxlan1

> ip link set dev veth1 netns my-overlay

set veth1 to network namespace my-overlay

```
docker@test1:~$ sudo ip netns exec my-overlay ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT group default ql
en 1000
    link/sit 0.0.0.0 brd 0.0.0.0
3: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEF
AULT group default qlen 1000
    link/ether 86:15:25:e3:dc:01 brd ff:ff:ff:ff:ff:ff
7: veth1@if6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT gr
oup default qlen 1000
    link/ether 76:74:84:29:a5:70 brd ff:ff:ff:ff:ff:ff
8: vxlan1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 sta
te UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 86:15:25:e3:dc:01 brd ff:ff:ff:ff:ff:ff
```

> ip netns exec my-overlay ip link set veth1 master br0
>
> ip netns exec my-overlay ip link set veth1 up

set veth1 as the slave of br0 and bring it up

> ip netns exec my-overlay ip neighbor add 10.0.10.10 lladdr $macaddr dev vxlan1

```
#upate arp(on host1 and host2) with

sudo ip netns exec my-overlay ip neighbor add 10.0.10.10 lladdr 92:1E:A1:74:BB:4D
dev vxlan1
sudo ip netns exec my-overlay ip neighbor add 10.0.10.11 lladdr 0A:42:35:1E:C2:E4
dev vxlan1
```

> But I still can't ping container c1 from c2 after this. Fix the problems to make it work.

after arp table is updated, we know that 10.0.10.11 is on mac address 0A:42:35:1E:C2:E4, but the computer don't know how to forward the package. So we still have to update the forward database(fdb).

```
docker@test1:~$ sudo ip netns exec my-overlay bridge fdb add 0A:42:35:1E:C2:E4 dev
 vxlan1 self dst 192.168.99.110 vni 42 port 4789
docker@test1:~$ docker exec c1 ping 10.0.10.11
PING 10.0.10.11 (10.0.10.11): 56 data bytes
64 bytes from 10.0.10.11: seq=0 ttl=64 time=0.479 ms
64 bytes from 10.0.10.11: seq=1 ttl=64 time=0.551 ms
64 bytes from 10.0.10.11: seq=2 ttl=64 time=3.389 ms

docker@test2:~$ docker exec c2 ping 10.0.10.10
PING 10.0.10.10 (10.0.10.10): 56 data bytes
64 bytes from 10.0.10.10: seq=0 ttl=64 time=0.483 ms
64 bytes from 10.0.10.10: seq=1 ttl=64 time=0.475 ms
64 bytes from 10.0.10.10: seq=2 ttl=64 time=0.475 ms
64 bytes from 10.0.10.10: seq=3 ttl=64 time=0.472 ms
```

we configured the fdb, telling the computer the Mac address 0A:42:35:1E:C2:E4 can be reached using vxlan1 interface, on host 192.168.99.110.

192.168.99.110 is the host of c2.

the fdb of host1 and host2 is as follows:

```
docker@test1:~$ sudo ip netns exec my-overlay bridge fdb show
33:33:00:00:00:01 dev br0 self permanent
01:00:5e:00:00:01 dev br0 self permanent
33:33:ff:bf:5d:75 dev br0 self permanent
92:1e:a1:74:bb:4d dev veth1
d6:5a:fd:38:9d:a8 dev veth1 permanent
33:33:00:00:00:01 dev veth1 self permanent
01:00:5e:00:00:01 dev veth1 self permanent
33:33:ff:38:9d:a8 dev veth1 self permanent
fe:27:46:77:7b:67 dev vxlan1 permanent
0a:42:35:1e:c2:e4 dev vxlan1
0a:42:35:1e:c2:e4 dev vxlan1 dst 192.168.99.110 self permanent


docker@test2:~$ sudo ip netns exec my-overlay bridge fdb show
33:33:00:00:00:01 dev br0 self permanent
01:00:5e:00:00:01 dev br0 self permanent
33:33:ff:cc:f3:2a dev br0 self permanent
0a:42:35:1e:c2:e4 dev veth1
f6:92:30:a7:f1:9c dev veth1 permanent
33:33:00:00:00:01 dev veth1 self permanent
01:00:5e:00:00:01 dev veth1 self permanent
33:33:ff:a7:f1:9c dev veth1 self permanent
92:1e:a1:74:bb:4d dev vxlan1
ca:c6:d0:22:d3:52 dev vxlan1 permanent
92:1e:a1:74:bb:4d dev vxlan1 dst 192.168.99.111 self   #this is learning by config
uring the "learning" option in vxlan creation.
```

the package transmission path is like:

veth2 -> veth1 -> br0 -> vxlan1 -> host2 -> vxlan1 -> br10 -> veth1 -> veth2