**source模式：**只能和指定的mac地址通信

**passthru模式：**会接管父设备（网口），父设备不能接收数据包，并且每个父设备只允许存在一个macvlan设备

passthrough模式只允许单个子接口连接主接口，且主接口必须设置成混杂模式，一般用于子接口桥接和创建vlan子接口的场景。

# macvlan内核代码分析

//kernel5.1.57

## 【1】macvlan代码分析

enum macvlan\_mode {

MACVLAN\_MODE\_PRIVATE = 1, /\* don't talk to other macvlans \*/

MACVLAN\_MODE\_VEPA = 2, /\* talk to other ports through ext bridge \*/

MACVLAN\_MODE\_BRIDGE = 4, /\* talk to bridge ports directly \*/

MACVLAN\_MODE\_PASSTHRU = 8,/\* take over the underlying device \*/

MACVLAN\_MODE\_SOURCE = 16,/\* use source MAC address list to assign \*/

};

vlan\_source\_hash: 只有MACVLAN\_MODE\_SOURCE模式才会用到

vlan\_hash: 与lower\_dev关联的所有macvlan虚拟口

//注册发包接口

static const struct net\_device\_ops macvlan\_netdev\_ops = {

.ndo\_start\_xmit = macvlan\_start\_xmit,

}

//注册收包接口

macvlan\_init\_module

->macvlan\_link\_register(&macvlan\_link\_ops);

static struct rtnl\_link\_ops macvlan\_link\_ops = {

.newlink = macvlan\_newlink,

}

->macvlan\_common\_newlink(...struct net\_device \*dev...)

{

struct macvlan\_port \*port;

//如果lowerdev上从未创建过macvlan虚拟口，则创建之

//并注册macvlan\_handle\_frame

if (!netif\_is\_macvlan\_port(lowerdev)){

macvlan\_port\_create(lowerdev);

-> netdev\_rx\_handler\_register(dev, macvlan\_handle\_frame, port);

}

}

//发包接口

macvlan\_start\_xmit

->macvlan\_queue\_xmit

->macvlan\_hash\_lookup

//收包接口

netif\_receive\_skb\_internal

->\_\_netif\_receive\_skb

->\_\_netif\_receive\_skb\_core //dev.c

{

rx\_handler = rcu\_dereference(skb->dev->rx\_handler);

}

->rx\_handler(&skb)

->macvlan\_handle\_frame

{

struct net\_device \*dev;

const struct macvlan\_dev \*vlan;

//MACVLAN\_MODE\_SOURCE

if (macvlan\_forward\_source(skb, port, eth->h\_source))

return RX\_HANDLER\_CONSUMED;

if (macvlan\_passthru(port)) //MACVLAN\_MODE\_PASSTHRU

vlan = list\_first\_or\_null\_rcu(&port->vlans,

struct macvlan\_dev, list);

else //其他模式

vlan = macvlan\_hash\_lookup(port, eth->h\_dest);

if (!vlan || vlan->mode == MACVLAN\_MODE\_SOURCE)

return RX\_HANDLER\_PASS;

dev = vlan->dev;

skb->dev = dev;

...

ret = NET\_RX\_SUCCESS;

handle\_res = RX\_HANDLER\_ANOTHER;

return handle\_res;

}

## 【2】先处理vlan，还是macvlan？

在老内核中，像处理普通协议一样处理vlan：先处理网桥，然后是vlan

static struct packet\_type vlan\_packet\_type \_\_read\_mostly = {

.type = cpu\_to\_be16(ETH\_P\_8021Q),

.func = vlan\_skb\_recv, /\* VLAN receive method \*/

};

dev\_add\_pack(...)

netif\_receive\_skb

{

if (skb->vlan\_tci && vlan\_hwaccel\_do\_receive(skb)){

//vlan硬件加速

}

//遍历ptye\_all链表, 上面的paket\_type.type为 ETH\_P\_ALL

list\_for\_each\_entry\_rcu(ptype, &ptype\_all, list){

...

}

skb = handle\_bridge(skb, &pt\_prev, &ret, orig\_dev);

skb = handle\_macvlan(skb, &pt\_prev, &ret, orig\_dev);

...

list\_for\_each\_entry\_rcu(ptype, &ptype\_base[ntohs(type) & PTYPE\_HASH\_MASK], list){

//此时才有可能处理vlan协议

}

}

-----------------------------------------------------------------------------------------------------------

新内核中，先处理vlan，再处理网桥

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sll\_pkttype为包类型. 可用的有:

PACKET\_HOST类型用于本机地址的包;

PACKET\_BROADCAST类型用于物理广播;

PACKET\_MULTICAST类型用于物理组播;

PACKET\_OTHERHOST用于在网卡混杂模式下从别的主机通信上接收包;

PACKET\_OUTGOING类型用于从本机packet socket发出的包

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_\_netif\_receive\_skb\_core //dev.c

{

list\_for\_each\_entry\_rcu(ptype, &skb->dev->ptype\_all, list)

if (skb\_vlan\_tag\_present(skb)){

if (vlan\_do\_receive(&skb)){

goto another\_round;

}

}

rx\_handler = rcu\_dereference(skb->dev->rx\_handler);

//vlan报文找不到vlan\_dev, 又不能被rx\_handler(例如macvlan)处理，则走下列分支

if (unlikely(skb\_vlan\_tag\_present(skb)) && !netdev\_uses\_dsa(skb->dev)) {

if (skb\_vlan\_tag\_get\_id(skb)){

skb->pkt\_type = PACKET\_OTHERHOST;

}else if(...){...}

}

list\_for\_each\_entry\_rcu(ptype, &ptype\_all, list)

}

## 【3】根据net\_namespace找路由信息

//ip\_input.c

ip\_rcv

{

struct net \*net = dev\_net(dev); //根据dev找自己的net\_namespace

skb = ip\_rcv\_core(skb, net);

}

ip\_rcv\_finish(struct net \*net, ...)

ip\_rcv\_finish\_core(net, sk, skb, dev, NULL);

ip\_route\_input\_noref

ip\_route\_input\_slow

fib\_lookup(net, &fl4, res, 0);

# 某场景某人设计的macvlan组网

host2: 目标机，拓扑我也没有整明白

eth0 eth1

| |

|————bond0—————|

| |

bond0.401 bond0.trunk(macvlan passthru)

| |

br401 br0

| |

vm(eth0) vm(eth1)

【】测试机

cat /etc/netplan/00-installer-config.yaml

network:

ethernets:

enp0s3:

dhcp4: true

enp0s8: {}

enp0s9: {}

bonds:

bond0:

interfaces:

- enp0s8

- enp0s9

parameters:

mode: active-backup

version: 2

modinfo bonding

cat /proc/net/bonding/bond0

ifconfig enp0s8 promisc

ifconfig enp0s9 promisc

ip link add link bond0 name bond0.401 type vlan id 401

ip link add link bond0 name bond0.401 type vlan id 1800

ip addr add 10.212.32.2/24 dev bond0.401

ip addr add 192.168.0.2/24 dev bond0.1800

ip link set bond0.401 up

ip link set bond0.1800 up

【】目标机

cat /etc/netplan/00-installer-config.yaml

network:

ethernets:

enp0s3:

dhcp4: true

enp0s8: {}

enp0s9: {}

bonds:

bond0:

interfaces:

- enp0s8

- enp0s9

parameters:

mode: active-backup

version: 2

modinfo bonding

cat /proc/net/bonding/bond0

ifconfig enp0s8 promisc

ifconfig enp0s9 promisc

(1) qbr401

ip link add qbr401 type bridge

ip netns add red

ip link add vnet1 type veth peer name tap1

ip link set tap1 netns red name eth0

ip netns exec red ip link set eth0 up

ip netns exec red ip addr add 10.212.32.1/24 dev eth0

ip link set vnet1 up

ip link add link bond0 name bond0.401 type vlan id 401

ip link set bond0.401 up

brctl addif qbr401 bond0.401

brctl addif qbr401 vnet1

ip link set qbr401 up

(2) bond0.trunk

ip link add link bond0 name bond0.trunk type macvlan mode passthru

ip link set bond0.trunk netns red name eth1

ip netns exec red ip link set eth1 up

ip netns exec red ip link add link eth1 name eth1.1800 type vlan id 1800

ip netns exec red ip addr add 192.168.0.1/24 dev eth1.1800

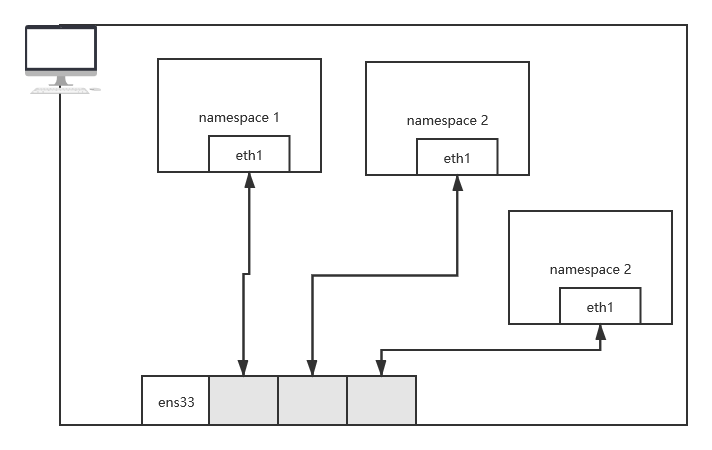
测试：

ip netns exec red ping 10.212.32.2

ip netns exec red ping 192.168.0.2

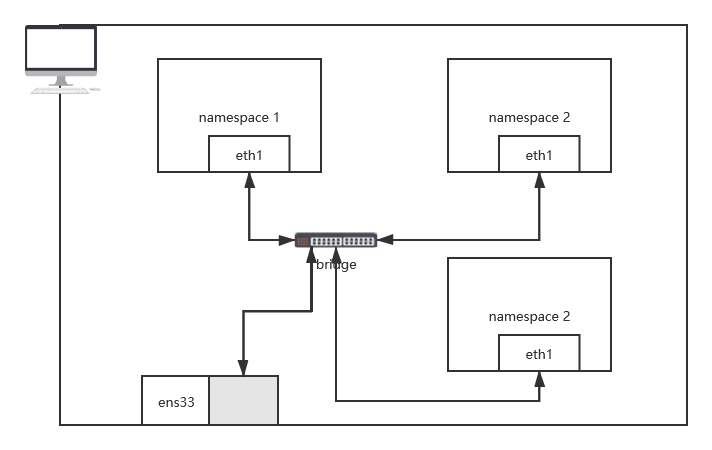
https://www.cnblogs.com/4a8a08f09d37b73795649038408b5f33/p/12200769.html

# macvlan几种模式

**vepa模式：**各个子设备直接无法直接通信（可以通过支持端口聚合的交换机通信），可以和外部通信。

**private模式：**和vepa模式类似，各个子设备之间无法通信，即使通过支持端口聚合的交换机也不能。

**bridge模式：**各个子设备之间可以通信，图里的桥是不存在的，画上仅仅为了便于理解



**source模式：**只能和指定的mac地址通信

**passthru模式：**会接管父设备（网口），父设备不能接收数据包，并且每个父设备只允许存在一个macvlan设备

