1. Virtual Bridge

Bridge is a link layer device and understands link layer protocols and therefore copiesdata frame by frame. In Linux, a bridge is a virtual device. As such, it cannot receive or transmit anythingunless you bind one or more real devices to it.

1. Bridge in linux

Opts of bridge

SIOCBRADDBR create a bridge

SIOCBRDELBRdelete a bridge

SIOCBRADDIF add an interface to bridge

SIOCBRDELIF delete an interface to bridge

1. SIOCBRADDBR

sudo brctl addbr br0 #add br0 bridge in on the host

or

int addBridge(const char \*brname)

{

int fd;

if (unlikely(brname == NULL))

return -1;

if((fd = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Error socket :%m\n", errno);

return -1;

}

if(ioctl(fd, SIOCBRADDBR, brname) < 0)

{

printf("Error add bridge %s :%m\n", brname, errno);

close(fd);

return -1;

}

close(fd);

return 0;

}

sock\_ioctl-🡪 br\_ioctl\_deviceless\_stub-🡪 br\_add\_bridge(struct net \*net, const char \*name)

{

//allocate a net\_device for bridge and init it through br\_dev\_setup . Then register it

dev = alloc\_netdev(sizeof(struct net\_bridge), name, br\_dev\_setup);

/\* dev->netdev\_ops = &br\_netdev\_ops; in the br\_dev\_setup\*/

res = register\_netdev(dev);

}

1. SIOCBRDELBR

sudo brctl delbr br0 #del br0 bridge in on the host

or

int delBridge(const char \*brname)

{

int fd;

if (unlikely(brname == NULL))

return -1;

if((fd = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Error socket :%m\n", errno);

return -1;

}

if(ioctl(fd, SIOCBRDELBR, brname) < 0)

{

printf("Error del bridge %s :%m\n", brname, errno);

close(fd);

return -1;

}

close(fd);

return 0;

}

sock\_ioctl-🡪 br\_ioctl\_deviceless\_stub-🡪br\_del\_bridge(struct net \*net, const char \*name)

{

//get a net\_devicefrom name and init it through br\_dev\_setup . Then register it

dev = \_\_dev\_get\_by\_name(net, name);

if (dev->flags & IFF\_UP) {

/\* Not shutdown yet. \*/

ret = -EBUSY;

}

else

br\_dev\_delete(dev, NULL);}

}

1. SIOCBRADDIF

sudo brctl addif br0 eth0#add eth0 to bridge br0

or

int bridgeAddPort(const char \*ifname, const char \*brname)

{

int s;

if (unlikely(ifname == NULL) || unlikely(brname == NULL))

return -1;

if((s = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Error up %s :%m\n", ifname, errno);

return -1;

}

struct ifreq ifr;

strcpy(ifr.ifr\_name, brname);

if (!(ifr.ifr\_ifindex = if\_nametoindex(ifname)))

{

printf("Error get %s ifindex :%m\n", ifname, errno);

close(s);

return -1;

}

if(ioctl(s, SIOCBRADDIF, &ifr) < 0)

{

printf("Error add %s to br0 :%m\n", ifname, errno);

close(s);

return -1;

}

close(s);

return 0;

}

sock\_do\_ioctl-🡪dev->netdev\_ops ->ndo\_do\_ioctl(br\_dev\_ioctl)-🡪 br\_add\_if(struct net\_bridge \*br, struct net\_device \*dev)

{

/\* No bridging of bridges \*/

if (dev->netdev\_ops->ndo\_start\_xmit == br\_dev\_xmit)

return -ELOOP;

/\* Device is already being bridged \*/

if (br\_port\_exists(dev))

return -EBUSY;

/\*enable promisc for the eth0 device\*/

dev\_set\_promiscuity(dev, 1);

/\*allocate a bridge port\*/

struct net\_bridge\_port \*p = new\_nbp(br, dev);

/\* insert the port to the datebase\*/

br\_fdb\_insert(br, p, dev->dev\_addr, 0); // the relevant fdb->is\_local set 1

/\*set receive hander to br\_handle\_frame:dev->rx\_handler= br\_handle\_frame\*/

netdev\_rx\_handler\_register(dev, br\_handle\_frame, p);

}

1. SIOCBRDELIF

sudo brctl delif br0 eth0 #add eth0 to bridge br0

or

int bridgeDelPort(const char \*ifname, const char \*brname)

{

int s;

if (unlikely(ifname == NULL) || unlikely(brname == NULL))

return -1;

if((s = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

{

printf("Error up %s :%m\n", ifname, errno);

return -1;

}

struct ifreq ifr;

strcpy(ifr.ifr\_name, brname);

if (!(ifr.ifr\_ifindex = if\_nametoindex(ifname)))

{

printf("Error get %s ifindex :%m\n", ifname, errno);

close(s);

return -1;

}

if(ioctl(s, SIOCBRDELIF, &ifr) < 0)

{

printf("Error add %s to br0 :%m\n", ifname, errno);

close(s);

return -1;

}

close(s);

return 0;

}

sock\_do\_ioctl-🡪dev->netdev\_ops ->ndo\_do\_ioctl(br\_dev\_ioctl)-🡪br\_del\_if(struct net\_bridge \*br, struct net\_device \*dev)

1. Frame handle in Bridge
2. netif\_receive\_skb(struct sk\_buff \*skb)

{

/\* the packet always deliver to the ptype\_all handler\*/

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

if (!ptype->dev || ptype->dev == skb->dev) {

if (pt\_prev)

ret = deliver\_skb(skb, pt\_prev, orig\_dev);

pt\_prev = ptype;

}

}

/\*handle the device belong to a bridge\*/

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

if (rx\_handler) {

switch (rx\_handler(&skb)) { /\* br\_handle\_frame is the handle\*/

case RX\_HANDLER\_CONSUMED:

ret = NET\_RX\_SUCCESS; /\*handle by bridge so it will not deliver to others Then out\*/

goto unlock;

case RX\_HANDLER\_ANOTHER:

goto another\_round;

case RX\_HANDLER\_EXACT:

deliver\_exact = true;

case RX\_HANDLER\_PASS:

break;

default:

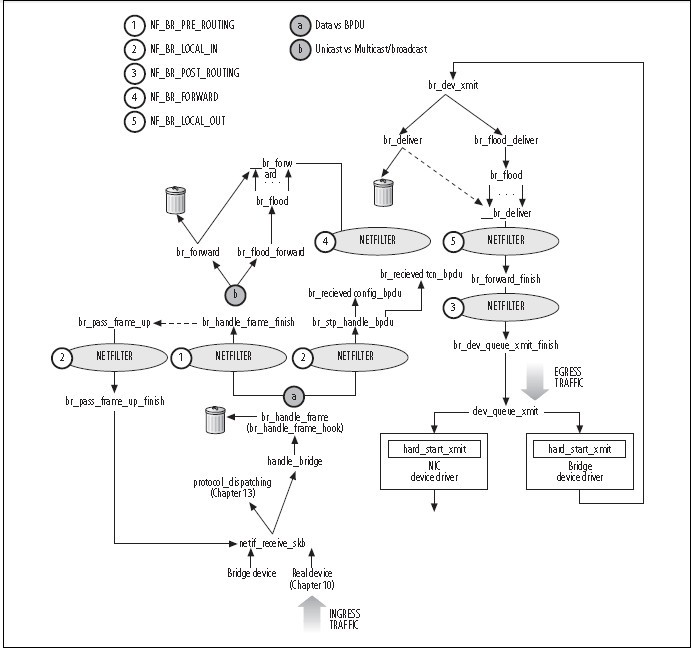
BUG();

}

}

}

1. Big Map of handler



intbr\_handle\_frame(struct sk\_buff \*skb)

{

/\*get bridge port of the net\_device\*/

p = br\_port\_get\_rcu(skb->dev);

if (ether\_addr\_equal(p->br->dev->dev\_addr, dest))/\*dest is equal to the bridge mac. So it shold send to the host\*/

skb->pkt\_type = PACKET\_HOST;

NF\_HOOK(NFPROTO\_BRIDGE, NF\_BR\_PRE\_ROUTING, skb, skb->dev, NULL,

br\_handle\_frame\_finish);

}

int br\_handle\_frame\_finish(struct sk\_buff \*skb)

{

/\*update fdatabase to record which port the packet’s sender belong to\*/

br\_fdb\_update(p->br, p, eth\_hdr(skb)->h\_source, vid);

/\*get the dst route from the fdatabase according to dest mac \*/

if (dst = \_\_br\_fdb\_get(br, dest, vid)) &&dst->is\_local) {

skb2 = skb;/\*if dst is local that means the packet is to the host\*/

/\* Do not forward the packet since it's local. \*/

skb = NULL;

}

if (skb)

{

if (dst) {

dst->used = jiffies;

/\*find the dst and forward it\*/

br\_forward(dst->dst, skb, skb2);

} else

/\*can not find then flood forward it\*/

br\_flood\_forward(br, skb, skb2);

}

if (skb2) /\*send to the host\*/

return br\_pass\_frame\_up(skb2);

}

int br\_forward(const struct net\_bridge\_port \*to, struct sk\_buff \*skb)

{

/\*send the packet through the device of to port\*/

skb->dev = to->dev;

skb\_push(skb, ETH\_HLEN);

br\_drop\_fake\_rtable(skb);

dev\_queue\_xmit(skb);

}

Int br\_pass\_frame\_up(struct sk\_buff \*skb)

{

/\*change the device to bridge device and send back to the netif\_receive\_skb \*/

skb->dev = brdev;

return NF\_HOOK(NFPROTO\_BRIDGE, NF\_BR\_LOCAL\_IN, skb, indev, NULL,

netif\_receive\_skb);

}

1. Some tips in practice
2. If a Packet receive from the net\_devcie belong to a bridge(bridge packets), the packet will go into the bridge handle.
3. The Dest Mac address of the bridge packets is equal the Mac addr of one of the port(net\_device belong to the bridge) of the bridge(including bridge itself). The packet should be the HOST\_PACKET and deliver back to the netif\_receive\_skb.

1) SRC IP: the packet only can be receive when the src ip is in the sub network of the bridge device and can’t be equal to any ip of host device./\* for routing reason\*/

2) DEST IP: des ip can be any ip of host device(including bridge device), Then the socket in userspace can receive it. It means the packet should for one of the interface aha.

1. If the bridge packets is not in the b situation, the packet will be unique forward or flood forward.