Linux下的虚拟Bridge实现

2013-04-21 23:28 by zmkeil, 3964 阅读, 2 评论, 收藏, 编辑

Linux下的Bridge也是一种虚拟设备，这多少和vlan有点相似，它依赖于一个或多个从设备。与VLAN不同的是，它不是虚拟出和从设备同一层次的镜像设备，而是虚拟出一个高一层次的设备，并把从设备虚拟化为端口port，且同时处理各个从设备的数据收发及转发，再加上netfilter框架的一些东西，使得它的实现相比vlan复杂得多。

1.Bridge的功能框图

它是Linux下虚拟出来bridge设备，Linux下可用brctl命令创建br设备，如下

brctl addbr brname

然后添加port，并进行相应配置，就可以使用了

brctl addif brname eth0

brctl addif brname eth1

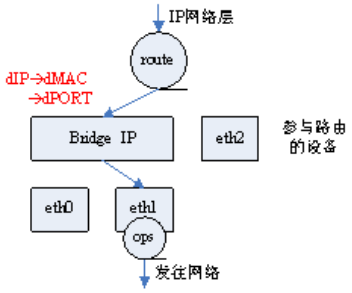
ifconfig brname IP up

ifconfig eth0 0.0.0.0 up

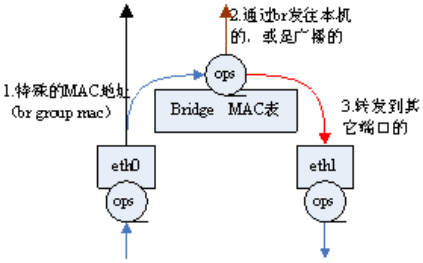
ifconfig eth1 0.0.0.0 up

可见br设备是建立在从设备之上的（这些从设备可以是实际设备，也可以是vlan设备等），并且可以为br准备一个IP（br设备的MAC地址是它所有从设备中最小的MAC地址），这样该主机就可以通过这个br设备与网络中的其它主机通信了（详见发送功能框图）。

另外它的从设备被虚拟化为端口port，它们的IP及MAC都不再可用，且它们被设置为接收任何包，最终由bridge设备来决定数据包的去向：接收到本机、转发、丢弃（详见接收功能框图）。



发送功能



接收功能

About

昵称：zmkeil
园龄：3年3个月
粉丝：37
关注：0
+加关注

SEARCH

最新评论

Re:Luci实现框架
您好，想请教一个问题，我想将Luci的admin-full下面的syslog显示功能移植到admin-mini，请问怎么实现？ -- zyzferrari

日历

< 2013年4月 >

日	一	二	三	四	五	六
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

随笔档案

- 2016年5月(2)
- 2016年2月(1)
- 2015年11月(1)
- 2015年2月(1)
- 2015年1月(1)
- 2013年8月(3)
- 2013年5月(9)
- 2013年4月(13)

随笔分类

- Linux开发杂记(4)
- 编程语言C/C++/JAVA(5)
- 操作系统(4)
- 计算机架构(1)
- 算法(2)
- 网络相关(15)
- 信号处理DSP(2)
- 有感而发(4)

推荐排行榜

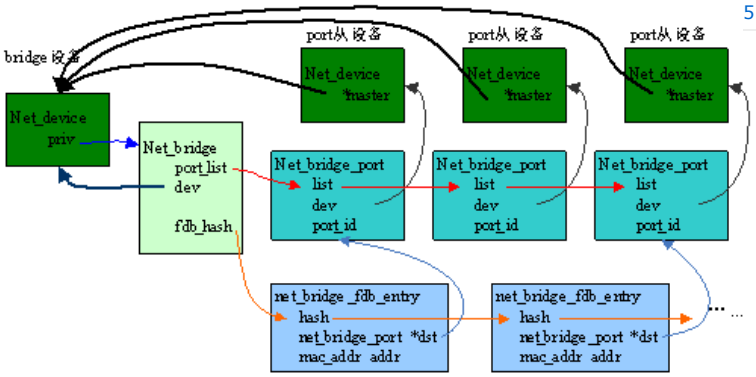
1. Linux下的虚拟Bridge实现(4)
2. 网络嵌入式设备(2)
3. 关于uC/OS的简单学习(2)
4. Luci实现框架(2)
5. uhttpd的实现框架(2)

阅读排行榜

1. Luci实现框架(12752)
2. uhttpd的实现框架(4069)

3. Linux下的虚拟Bridge实现(3963)
4. OpenWRT平台搭建及简单应用(3162)
5. Linux下VLAN功能的实现(1967)

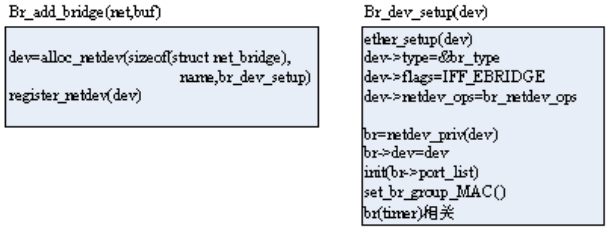
简单的数据结构框图如下所示。



2.Bridge设备的创建

和Vlan一样，bridge也被当成一个module加载进内核，它的module_init()函数和vlan差不多，进行一些namespace的注册，特殊的是它还注册了一个netfilter_ops，在内核全局的HOOK函数表中增加了7个函数，其中5个的pf=Bridge，另两个的pf分别为INET、INET6，它们主要用于bridge中的netfilter操作（后面会细讲）。

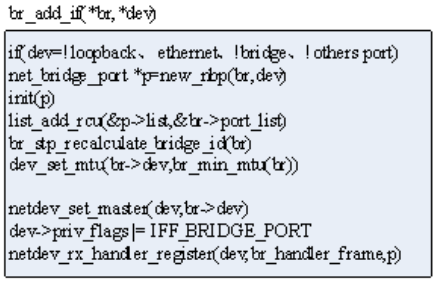
最后，也是我们这最关心的是，它注册了一个ioctl函数br_ioctl_deviceless_stub()，该ioctl函数和vlan的一样，都会作为sock_ioctl()的特殊情况被调用。映射到应用层，它应该是对某个socket插口进行ioctl操作，详见brctl源码。该ioctl函数中最主要的就是br_add_bridge(net,buf)，用于创建bridge设备，如下图所示：



该函数调用netdev_alloc()，申请net_device()，并分配私有空间net_bridge()结构，指明初始化函数为br_dev_setup()，最后register_netdev()把该设备组册进内核，可见bridge设备和一般的设备差不多。

主要看其中br_dev_setup()，首先初始化设备的type、flags为bridge，然后最关键的是设置其dev->netdev_ops = br_netdev_ops，即内核为bridge设备准备好了一套通用的驱动函数，这个直接关系到bridge的工作方法，后面再细讲。然后初始化私有空间net_bridge()结构，设置bridge的本地设备及从设备的list（当然这是还没有从设备加进来），然后设置了桥的group_address，即上一节所说的特殊的MAC地址，最后还初始化了timer相关的。

这时bridge还不完整，还需添加port从设备，由命令brctl addif brname portdev完成，但要注意，虽然还是brctl命令，但此时的操作对象是已经存在的bridge设备，映射到内核中就是br_netdev_ops->ioctl()中的br_add_if()（它是br设备的ioctl操作，和之前那个sock_ioctl的分支不是一个层次上的）。至于怎么从应用层直接操作底层的net_device设备的，可以参见brctl源码，以后再看吧，先看看这里的br_add_if()，如下图：

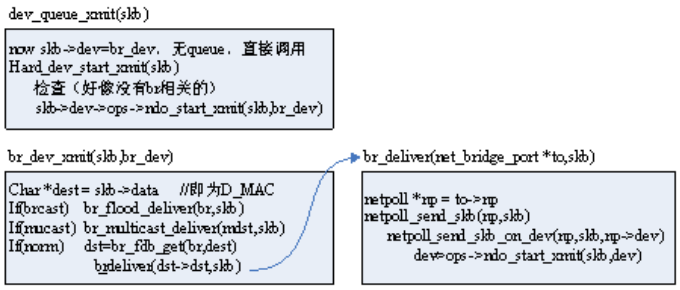


首先判断dev从设备必须不是loopback，不是bridge，不是其他bridge的port，且要是ethernet设备，才能继续；然后根据br、dev选择一个index号，并分配一个新的net_bridge_port结构，初始化之，并将它加入bridge的port_list中；最后br的一些物理参数，其MAC地址为所有从设备中MAC最小的（由上一节知，从设备被设置成全接收模式，其IP和MAC都没有了），且其MTU也为所有从设备中最小的。

上面设置br的相关参数，下面还要设置从设备，首先使dev->master=br_dev（实际上就是构成上一节数据结构中的索引关系）；然后设置dev->prive_flags加上IFF_BRIDGE_PORT，这样它就不能再作为其他br的从设备了；最后也是最关键的，设置dev->rx_handler为br_handler_frame()，为数据接收作准备。

3.Bridge设备的发送流程

前面也讲过了，Linux下的bridge设备，对下层而言是一个桥设备，进行数据的转发（实际上对下也有接收能力，下一节讲）。而对上层而言，它就像普通的ethernet设备一样，有自己的IP和MAC地址，那么上层当然可以把它加入路由系统，并利用它发送数据啦，并且很容易想到，它的发射函数最终肯定是利用某个从设备的驱动去完成实际的发送的，这个和VLAN是相通的。具体看代码：

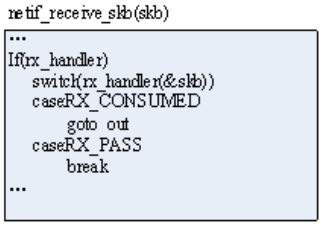


上层根据目的IP地址，路由选择了该br_dev设备发送，并且由ARP缓存中得到了对应的目的MAC，填写在了skb中，然后启动了发送流程dev_queue_xmit(skb)。因为此时的skb->dev为br_dev，无queue，直接去调用br设备的发送函数，该函数就是br_netdev_ops中定义的br_dev_xmit(skb,br_dev)。

该函数首先根据目的MAC地址，确定是广播还是单播，这里仅讨论单播时，根据DMAC在net_bridge的fdb_hash中找到相应的net_bridge_fdb_entry项，并索引到对应的端口net_bridge_port。最后利用该端口的从设备来发送数据，注意，这里是直接调用dev->ops->ndo_start_xmit(skb,dev)的，一放面这里的dev已经是从设备了，另一方面，这里没有像VLAN中那样重定位skb->dev，并重启发送流程dev_queue_xmit()，是因为一个从设备只能作为一个bridge的port，没有其它身份，不存在竞争问题。

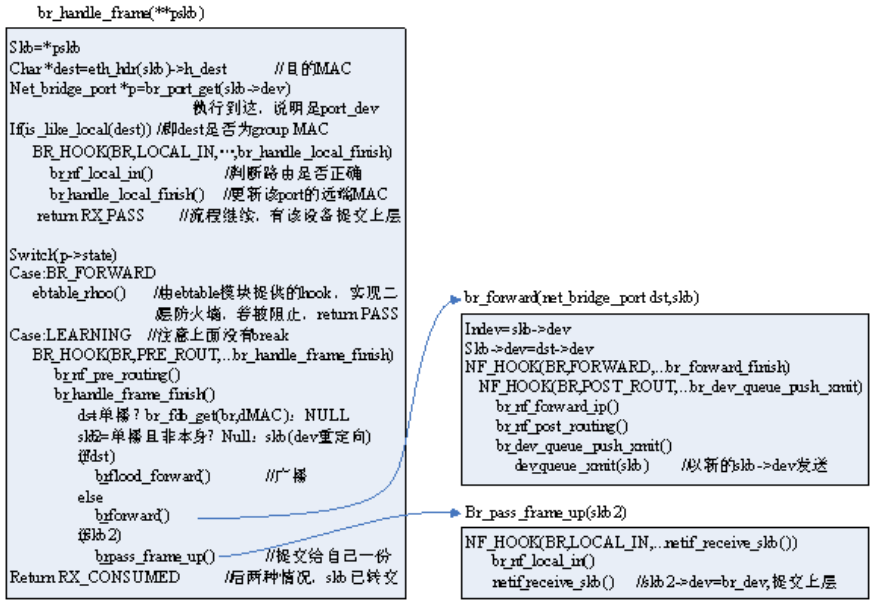
4.Bridge设备的接收流程

和VLAN一样，实际接收由硬件设备完成，最终通过netif_receive_skb(skb)函数提交给上层，而在该函数中会处理vlan、bridge这类特殊设备。与LVAN的仅是把skb设备重定位以实现对上层透明的要求不同，Bridge接受过程复杂得多，因此专门注册了一个函数来处理，即前面提到的rx_handler()，它被注册在port设备的net_device结构中（这算是port设备失去自身IP、MAC的一个补偿吧◎），如下图所示。只有作为bridge的从设备才会注册rx_handler()，并在这里执行，处理桥接，普通的设备不会执行到这里。



这里两种典型的返回值是RX_CONSUMED、RX_PASS，后者表示处理了一下回来，继续之前的流程，实际上就是对应的接收功能框图中的第一种情况；前者表示该skb已不再属于这个从设备了，而是被提交给了br设备，所以本次netif_receive_skb()就不用管啦，直接goto out，这里还要再分两种情况，一是转发的，这时br就真的充当了桥的角色，二是由br提交给上层的，这时br充当的是一个以太网设备，如前面所述。

要处理这么多情况，代码需设计得很巧妙，这里的rx_handler被设置为br_handle_frame(**pskb)，看具体代码：



功能框架清楚了，代码流程就清楚了，就不细看了，有几个注意的地方：一是bridge的端口处于FORWARD或LEARNING状态没多大区别，只是FORWARD要多执行一个二层防火墙，所有用了一个中间没有break的switch结构；二是要时刻记着，bridge本身除了有转发端口外，自己也是一个设备，广播（多播）时也要发一份给自己，且是以br_dev的身份递交上层；三是所有的HOOK函数都会比较复杂，因为内核的netfilter框架建立在网络层，而bridge在链路层就转发了，相当于跳过了netfilter，所以在这些hook中都会去调用INET域的hook函数。

5.小结

通过对Bridge和vlan的学习，了解了网络栈底层的工作方式，发送这个主动过程相对简单，而接收过程则相对复杂，用到BH模型，NAPI等。

Vlan和bridge功能有所不同，但相似处很多，更重要的是：它们都对上层透明，所以不会牵扯到协议域的问题。

zmkeil
关注 - 0
粉丝 - 37
+加关注

4 0

« 上一篇：Linux下VLAN功能的实现
» 下一篇：由PPPOE看Linux网络协议栈的实现

分类：网络相关

#1楼 lemon_m
2014-03-24 11:14

ADD YOUR COMMENT

感谢博主的学习分享，的确让人获益良多。

支持(0) 反对(0)

#2楼 SammyLiu
2015-07-22 10:55

好文章

支持(0) 反对(0)

最新知识库文章：
· 程序猿媳妇儿注意事项
· 可是姑娘，你为什么要编程呢？
· 知其所以然（以算法学习为例）
· 如何给变量取个简短且无歧义的名字
· 编程的智慧
» 更多知识库文章...