Linux实现的IEEE 802.1Q VLAN

  (2012-07-04 17:47:28)

转载▼

|  |  |
| --- | --- |
| 标签： [杂谈](http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag) | 分类： [NET](http://blog.sina.com.cn/s/articlelist_1028986757_9_1.html) |

**第一部分：VLAN的核心概念**

说起IEEE 802.1q，都知道是VLAN，说起VLAN，基本上也没有盲区，网络基础。然而说到配置，基本所有人都能顺口溜一样说出Cisco或者H3C设备的配置命令，对于Linux的VLAN配置却存在大量的疑问。这些疑问之所以存在我觉得有两点原因：
**1.对VLAN的本质还是没有理解。**
不管你的Cisco/H3C命令敲得再熟练，如果看不懂Linux的vconfig，那么也将无法掩饰你对概念理解的浅显；
**2.对Linux实现虚拟网络设备风格不熟悉**

可能你已经十分理解802.1q了，也许还看过了IEEE的文档，然而却对Linux的Bridge，tap，bond等虚拟设备不是很理解，那么也将无法顺利配置VLAN。

对于VLAN概念的理解，有几点要强调：
**1.VLAN分离了广播域；
2.单独的一个VLAN模拟了一个常规的交换以太网，因此VLAN将一个物理交换机分割成了一个或多个逻辑交换机；
3.不同VLAN之间通信需要三层参与；
4.当多台交换机级联时，VLAN通过VID来识别，该ID插入到标准的以太帧中，被称作tag；
5.大多数的tag都不是端到端的，一般在上行路上第一个VLAN交换机打tag，下行链路的最后一个VLAN交换机去除tag；
6.只有在一个数据帧不打tag就不能区分属于哪个VLAN时才会打上tag，能去掉时尽早要去掉tag；
7.最终，IEEE 802.1q解决了VLAN的tag问题。除了IEEE 802.1q，其余的都是和实现相关的，虽然Cisco和H3C的实现很类似，Linux可以和它们有大不同。**
关键看最后3点，也就是3，4，5。这是VLAN最难理解的部分，不过一旦理解了，VLAN也就不剩下什么了。为了使得叙述上以及配置上更加的方便，Cisco以及其他的厂商定义了很多的细节，而这些细节在IEEE 802.1q标准中并没有被定义，这些细节包括但不局限于以下几点：

**1.每一个VLAN交换机端口需要绑定一个VLAN id；**

**2.每一个VLAN交换机端口处于下面三类中的一类：access，trunk，hybrid。**

**2.1.access端口：从此类端口收到的数据帧是不打tag的，从此类端口发出的数据帧是不打tag的；**

**2.2.trunck端口：从此类端口收到的数据帧打着tag，从此类端口发出的数据帧需要打tag(不考虑缺省VLAN的情况)；**

**2.3.hybrid端口：略**

我们实则没有必要去深究Cisco/H3C的命令以及到底那三类端口类型有何区别，之所以有三类端口类型完全是为了将VLAN的概念(最终的IEEE 802.1q标准)很方便的用起来。说白了，trunk端口的存在是因为不得已，因为有属于多个VLAN的数据帧要通过单一的物理链路，不打tag是无法区分各自属于哪个VLAN的，于是就有了IEEE 802.1q这个标准，定义了一个tag插入到以太帧中，为了使这个理论性的东西被使用起来，厂商便定义了一系列的概念性的东西，比如和tag相关的链路就是trunk链路之类。
于是乎，我们可以完全抛开任何的配置命令，抛开任何厂商定义的东西，完全按照IEEE 802.1q标准以及我们的需求来理解VLAN，这样下来之后，你绝对可以在Linux上完美实现任何VLAN配置了。首先我们定义一下我们的需求以及满足该需求的网络拓扑，关键看如何接线。

**1.情况一.同一VLAN内部通信**

**1.1.同一交换机同一VLAN的不同端口进行通信**



**1.2.不同交换机的不同端口进行通信**



**2.情况二.不同VLAN之间通信**

**2.1.同一交换机不同VLAN之间进行通信**



**2.2.不同交换机的不同VLAN进行通信**

从上述1.2可以看出，为了节省线缆和避免环路，两个VLAN交换机的两个端口之间的同一条链路需要承载不同的VLAN数据帧，为了使彼此能够识别每个数据帧到底属于哪个VLAN，十分显然的办法就是为数据帧打上tag，因此上述1.2中的端口J和端口K之间的链路上的数据帧需要打tag，端口J和端口K都同属于两个VLAN，分别为VLAN m和VLAN n。换句话说，只要一个端口需要传输和接收属于多个VLAN的数据帧，那么从该端口发出的数据帧就要打上tag，从该端口接收的数据帧可以通过tag来识别它属于哪个VLAN，用Cisco/H3C等厂商的术语来讲，它就是trunk端口，两个trunck端口之间的链路属于trunk链路。
我们知道，一般而言，我们的PC机直接连接在常规二层交换机或者支持VLAN的交换机端口上，而我们的PC机发出的一般都是常规的以太网数据帧，这些数据帧是没有tag的，它们可能根本不知道802.1q为何物，然而VLAN存在的目的就是把一些PC机划在一个VLAN中，而把另一些PC机划在另一个VLAN中从而实现隔离，那么很显然的一种办法就是将支持VLAN的交换机的某些端口划在一个VLAN，而另一些端口划在另一个VLAN中，一个VLAN的所有端口其实就形成了一个逻辑上的二层常规交换机，同属于一个VLAN的PC机连接在划在同一个VLAN的端口上，为了扩展VLAN，鉴于单台交换机端口数量的限制，需要级联交换机，那么级联链路上则同时承载着不同VLAN流量，因此级联链路则成为trunk链路，所有不是级联链路的链路都是直接链路，用厂商术语来讲就是access链路(注意，这里暂且不谈hybrid)，自然而然的，access链路两端的端口都是和tag无关的，只需要做到“没有tag直通，有tag去掉即可”，因此它可以连接PC机或者常规交换机以及VLAN交换机的非trunk端口。
VLAN的内容基本也就是以上那些了，分为三部分：

**1.设计目的**

隔离广播域，节省物理设备，隔离安全策略域

**2.IEEE 802.1q**

为扩展VLAN的级联方案提供了一个标准的协议

**3.如何使用VLAN**

将某些端口划为一个VLAN，基于MAC地址什么的...
其实，至于怎么划分VLAN，标准中并没有给出什么硬性的规定，只要能够保证属于同一VLAN的端口完全否则IEEE 802系列的标准即可，换句话说就是属于同一VLAN的所有交换机的所有同一VLAN的端口完全就是一个以太网即可，透传以太帧。
到此为止，我们基本上已经忘了配置trunk，access，基于端口划VLAN的命令了，脑子里面留下的只是VLAN的核心概念，使用这些核心的概念，我们就可以在Linux上配置完整的VLAN方案了，如果你去硬套Cisco的配置，那么结果只是悲哀。比如如果你问：如何在Linux上配置端口为access，如何在Linux上将某些网卡划到一个VLAN...
理解Linux Bridge的都知道，Linux本身就可以实现多个Bridge设备，因为Linux的Bridge是软的，所以一个Linux Box可以配置多个逻辑意义的Bridge，而多个Bridge设备之间必须通过第三层进行通信，加之第三层正是以太网的边界，因此一个Linux Box也就可以模拟多个以太网了，不同的Bridge设备就可以代表不同的VLAN。

**第二部分：Linux上的VLAN**

Linux上的VLAN和Cisco/H3C上的VLAN不同，后者的VLAN是现有了LAN，再有V，也就是说是先有一个大的LAN，再划分为不同的VLAN，而Linux则正好相反，由于Linux的Bridge设备是被创建出来的逻辑设备，因此Linux需要先创建VLAN，再创建一个Bridge关联到该VLAN，创建VLAN很简单：
**ifconfig eth0 0.0.0.0 up
vconfig eth0 10
ifconfig eth0.10 up**
当使用vconfig创建了eth0.10之后，它就是一个“真实意义”的虚拟网卡设备了，类似br0，tap0，bond0之类的，在这个虚拟网卡之下绑定的是一个真实网卡eth0，也就是数据从eth0这块真实网卡发出，eth0.10中的“.10”表示它可以承载VLAN 10的数据帧，并且在通过eth0发出之前要打上tag。那么打tag这件事自然而然就是通过eth0.10这个虚拟设备的hard\_xmit来完成的，在这个hard\_xmit中，打上相应的tag后，再调用eth0的hard\_xmit将数据真正发出，如下图所示：

因此一个真实的物理网卡比如ethx，它可以承载多个VLAN的数据帧，因此它就是trunk端口了，如下所示：

Linux的VLAN工具vconfig采用ethx.y的方式以ethx为trunk端口加入VLAN id为y的VLAN中。类比Cisco/H3C，我们已经创建了trunk，总结一下：使用vconfig创建一个ethx.y的虚拟设备，就创建了一个trunk，其中ethx就是trunk口，而y代表该trunk口连接的trunk链路可以承载的VLAN数据帧的id，我们创建ethx.a，ethx.b，ethx.c，ethx.d，就说明ethx可以承载VLAN a，VLAN b，VLAN c，VLAN d的数据帧。
接下来，我们看一下如何创建access端口。首先注意，由于Linux的Bridge是虚拟的，逻辑意义的，因此可以先创建了VLAN之后，再根据这个VLAN动态的创建Bridge，而不是“为每一个端口配置VLAN id”，我们需要做的很简单：
**创建VLAN：
ifconfig eth0 0.0.0.0 up
vconfig eth0 10
ifconfig eth0.10 up
为该VLAN创建Bridge：
brctl addbr brvlan10
brctl addif brvlan10 eth0.10
为该VLAN添加网卡：
ifconfig eth1 0.0.0.0 up
brctl addif brvlan10 eth1
ifconfig eth2 0.0.0.0 up
brctl addif brvlan10 eth2**
...
这就完了。从此，eth1和eth2就是VLAN 10的access端口了，而eth0则是一个trunk端口，级联VLAN的时候要用到，如果不需要级联VLAN，而仅仅需要扩展VLAN 10，那么你大可将eth1连接在一个二层常规交换机或者hub上...同样的，你可以再创建一个VLAN，同样通过eth0来级联上游VLAN交换机：
**ifconfig eth0 0.0.0.0 up
vconfig eth0 20
ifconfig eth0.20 up
brctl addbr brvlan20
brctl addif brvlan20 eth0.20
ifconfig eth5 0.0.0.0 up
brctl addif brvlan20 eth5**
如下图所示：

这下基本就搞定了Linux上VLAN的配置，接下来还有一个内容，那就是VLAN之间的通信。这个知识点最简单了，那就是使用路由，为此很多人把支持VLAN的三层交换机和路由器等同起来。既然使用路由就需要一个IP地址作为网关，那么如何能寻址到这个IP地址自然就是一个不可回避的问题，我们要把这个IP配置在哪里呢？可以肯定的是，必须配置在当前VLAN的某处，于是我们有多个地方可以配置这个IP：

**1.同属于一个VLAN的路由器接口上，且该路由器有到达目的VLAN的路由(该路由器接口为trunk口)。**

**2.同属于一个VLAN的ethx.y似的虚拟接口上，且该Linux Box拥有到指定VLAN a的路由(最显然的，拥有ethx'.a虚拟接口)。**

**3.同属于一个VLAN的Bridge设备上(Linux的Bridge默认带有一个本地接口，可以配置IP地址)，且该Linux Box拥有到指定VLAN a的路由(最显然的，拥有ethx'.a虚拟接口或者目标VLAN的Bridge设备)。**

其中的1和2实际上没有什么差别，本质上就是找一个能配置IP地址的地方，大多数情况下使用2，但是如果出现同一个VLAN在同一个Linux Box配置了两个trunk端口，那么就要使用Bridge的地址了，比如下面的配置：
**brctl addbr brvlan10
brctl addif brvlan10 eth0.10
brctl addif brvlan10 eth1.10
ifconfig brvlan10 up**
此时有两个ethx.y型的虚拟接口，为了不使路由冲突，只能配置一个IP，那么此IP地址就只能配置在brvlan10上了。不管配置在Bridge上还是配置在ethx.y上，都是要走IP路由的，只要MAC地址指向了本地的任意的一个接口，在netif\_receive\_skb调用handle\_bridge的时候都会将数据帧导向本地的IP路由来处理。Linux作为一个软件，其并没有原生实现硬件cache转发，因此对于Linux而言，所谓的三层交换其实就是路由。
我们看一下一个被打上tag的数据帧什么时候脱去这个tag，在定义上，它是从access端口发出时脱去的，然而在语义上，只要能保证access端口发出的数据帧不带有tag即可，因此对于何时脱去tag并没有什么严格的要求。在Linux的VLAN实现上，packet\_type的func作为一个第三层的处理函数来单独处理802.1q数据帧，802.1q此时和IP协议处于一个同等的位置，VLAN的func函数vlan\_skb\_recv正如IP的处理函数ip\_rcv一样。在Linux实现的VLAN中，只有当一个端口收到了一个数据帧，并且该数据帧是发往本地的时候，才会到达第三层的packet\_type的func处理，否则只会被第二层处理，也就是Bridge逻辑处理，Linux的原生Bridge实现并不能处理802.1q数据帧，甚至都不能识别它。整个trunk口收发数据帧，IEEE 802.1q帧处理，以及VLAN间通信的示意图如下：







到此为止，Linux的VLAN要点基本已经说完了，有了这些理解，我想设计一个单臂Linux Box就不是什么难事了，单臂设备最大的优势就是节省物理设备，同时还能实现隔离。这个配置不复杂，如果不想用VLAN实现的话也可以用ip addr add dev ...增加虚拟IP的方式来实现，然而用VLAN实现的好处在于可以和既有的三层交换机进行联动，也可以直接插在支持标准的IEEE 802.1q的设备的trunk口上。
机制搭台，策略唱戏。既然VLAN的实现机制已经了然于胸了，那么它的缺点估计你也看到了，如何去克服呢？PVLAN说实在的是一个VLAN的替代方案。解决了VLAN间的IP网段隔离问题，我们在Linux上如何实现它呢？这倒也不难，无非就是在LAN上添加一些访问控制策略罢了，完全可以用纯软件的方式来实现，甚至都可以用ebtables/arptables/iptables来实现一个PVLAN。如果说VLAN是一个硬实现的VLAN的话，那么PVLAN纯粹是一个软实现的VLAN，甚至都不需要划分什么VLAN，大家都处于一个IP网段，只需要配置好访问控制策略即可，使得同一IP子网的Host只能和默认网关通信，而之间不能通信，所以说，即使你不知道“隔离VLAN”，“团体VLAN”之类的术语，实际上你已经实现了一个PVLAN了。

**第三部分：几点总结**

1.你需要首先规划出你的网络拓扑而不是先去研究VLAN在Linux上如何配置以及如何实现；
2.你需要深入理解VLAN设计的初衷，该配置哪些东西；
3.你需要知道对于VLAN哪些概念是核心，哪些概念并不是必须的。
4.不管基于什么平台配置VLAN，只有两点是必须的：a.哪些端口属于哪个VLAN；b.哪个端口是级联端口，属于多个VLAN。
5.其它的都不用去死记硬背，都是浮云...

转载：<http://blog.csdn.net/dog250/article/details/7354590>