

# 自己动手打造一台路由器

刘 磊

---

# 自己动手打造一台路由器

刘 磊

版权 © 2013

---

# 目录

|  |    |
|--|----|
| 前言 .....   | iv |
| 1. 搭建运行环境 .....  | 1  |
| 安装Ubuntu .....   | 1  |
| 编译vde .....  | 1  |
| 安装qemu .....   | 3  |
| 2. 搭建网络环境 .....  | 6  |
| 搭建主机网络 .....   | 6  |
| 搭建桥接网络 .....   | 7  |
| 搭建VDE网络 .....  | 8  |
| slirpvde .....   | 9  |
| 3. 搭建嵌入式系统 .....   | 10 |
| 运行OpenWrt .....  | 10 |
| 运行buildroot .....  | 13 |
| 4. 设备集成 .....  | 17 |
| 组装路由器 .....  | 17 |
| lan侧桥接设置 .....   | 23 |
| 功能验证 .....   | 26 |
| 5. 防火墙篇 .....  | 28 |
| 准备工作 .....   | 28 |
| 基本防火墙 .....  | 28 |
| 目标描述 .....   | 29 |
| 创建交换机网络 .....  | 29 |
| 创建wan侧PC .....   | 30 |
| 创建路由器 .....  | 31 |
| 创建lan侧PC .....   | 32 |
| 创建防火墙规则 .....  | 33 |
| 6. 防火墙高级篇 .....  | 40 |
| 7. 虚拟私有网络 .....  | 41 |
| L2TP连接 .....   | 41 |
| 准备工作 .....   | 41 |
| 开始实验！ .....  | 45 |
| 创建IPSEC连接 .....  | 46 |
| 准备工作 .....   | 47 |
| 配置阶段 .....   | 47 |
| 8. Quality of Service (QOS) .....  | 50 |
| 准备工作 .....   | 50 |
| 实验阶段 .....   | 50 |
| TBF简单限速 .....  | 51 |
| A. 常见问题FAQ .....   | 53 |
| openwrt pppd 段错误 .....   | 53 |
| failed to initialize KVM: Device or resource busy No<br>accelerator found! ..... | 53 |
| 参考文献 .....   | 54 |

---

# 前言

关于写作此书的目的 -- 跳槽。

所以本书并不讨论高深的理论知识，而是把自己一年来关于网络相关的经验记录下来，以供新手上路。然而仅仅是记录一些经验，难免有些琐碎，不便于学习，所以我将会用实例来一步步把网络相关的知识介绍给大家。所以本书中不会介绍太多的理论知识，其实说来惭愧，我个人其实也是才疏学浅，让我完全给大家讲明白，也实在力不从心，所以本书将尽量用实例和脚本代码给大家娓娓道来。

本人所学专业 and 程序并无太大关系，只是处于兴趣才选择了这个行业，可是，现实总是残酷的，在学校里是出于兴趣，但到了工作上，只是出于兴趣是远远不够的，还要学会忍受没有兴趣的事，于是，渐渐的兴趣日益淡薄，没有动力，没了理想。

接下来，自然就有了跳槽的想法，但是其实心里也明白，跳槽其实也改变不了现状的，因为我是个Coder，而不是Creator，显然，老板更希望你能老老实实的工作，而不是整天胡思乱想而企图拯救世界人民于水火之中的，所以，心情甚是郁闷，然而公司氛围很好，一走了之显然太不仗义，也不是本人的风格，故作此书，希望能为公司尽微薄之力。

其实选择做程序员，并不是因为自己是理科出身，适合于此，记得高中时期，本人数学基本上都没及格过几次，其他理科也不尽人意，学历不高，天资也无过人之处，只是寄托个人的理想在里面。

本人是一个比较情绪化的人，喜欢做自己喜欢的事情，对于感兴趣的事物，倾其所有也不后悔，不感兴趣的事情则是不闻不问，毫不关心。其中自有利弊，酸甜苦辣，我也尽尝其味，正所谓，什么样的人，走什么样的路，虽然遇到不少曲折，或许这就是人生吧。

回忆往昔，其实让我永远铭记的也只有那么两件事情，一是足球，二是高中时期那让人难以忘怀的初恋（未果），虽然现如今已物是人非了（可能有点夸张了，我还是风华正茂的年龄），其实感觉都是一样的，那就是发自内心毫无掩饰的喜爱！如今，我不知道还能不能找到这种感觉了，那种痴迷，那种疯狂，那种“年少不知愁滋味”锋芒！

而如今，我在朝着成为一名好员工的道路上前进，不再感到厌烦，而是心平气和地工作，成为一个更听从领导安排，努力工作的，让老板喜欢的“好员工”，不会去表达自己的意愿，只是埋头苦干。因为我只是为了过活！

所以，趁着自己还有点力气的时候，写下此书，也算是个纪念吧。

---

# 第 1 章 搭建运行环境

既然我们要自己动手制作一个路由器，那么我们就需要一个开发/生产环境了。由于我们的路由器是运行在虚拟机中的，所以我们并不需要电线啊，电路板或者开发板之类的东西，况且他们还要耗费我们的银子。

## 安装Ubuntu

Ubuntu是linux中用户体验相当好的一款linux发行版本，至于说怎么好，这个就因人而异了，“萝卜青菜，各有所爱！”我使用Ubuntu有好多年了，用着很稳定也很方便，所以也很少尝试其他的发行版本了，所以这里就以Ubuntu作为运行系统，然而我却十分不适应Ubuntu的Unity界面，于是我们在来一个折中--Xubuntu，他是Ubuntu的一个衍生版本，本身并没有太大的区别，只是用户界面使用的是Xfce，相对来说界面更传统，资源占用率更小一些。当然这只是我个人的使用习惯而已，好了，镜像下载看这里：

- <http://xubuntu.org/getxubuntu/>

这里我们选择“Mirror downloads”，“Torrent downloads”也不错，然后选择一个镜像服务器即可，这里我选择“United States”，根据自己的需要选择合适的版本，这里我们选择：

- <http://mirror.anl.gov/pub/ubuntu-iso/CDs-Xubuntu/12.10/release/xubuntu-12.10-desktop-amd64.iso>

由于目前Xubuntu还没有国内的镜像，所以下载会比较慢，所以请耐心等待吧。

好了，下载OK，接下来我们开始安装系统了，这里有三种方式：

- 光盘安装
- U盘安装
- Wubi

上述3种安装方法网上有很多资源，大家可以谷歌一下，这里就不多介绍了，其中Wubi的方法比较独特，那就是他是把Ubuntu当作一个应用程序一样安装到Windows系统中的，好处是不用格式化硬盘了，只需要占用Windows的分区空间就行了，不过从效率上来说没有前两种方法好，如果你只是尝试一下的话，推荐使用Wubi。如果还嫌麻烦的话，那么就可以考虑使用虚拟机了。

系统安装完，我们还需要添加一些工具到系统中：

```
liunx@ubuntu:~$ sudo apt-get install build-essential
```

这里我们将安装基本的编译工具集合，包括编译器之类的工具，当然仅仅这样还不够，我们接下来还要安装一些相关的dev文件，这个我们会一步一步地添加的。

## 编译vde

ubuntu软件包里似乎对vde不是特别重视，更新的不是很及时，所以我们自己动手编译vde，首先我们通过svn来获取vde的源代码：

```
liunx@ubuntu:~/Work/Emulator/work$ sudo apt-get install subversion
liunx@ubuntu:~/Work/Emulator/work$ svn co https://vde.svn.sourceforge.net/svnroot/vde/
trunk/vde-2 vde_svn
```

ubuntu默认是没有安装svn的，所以需要安装，vde源码是用automake tools进行配置的，所以我们还需要安装automake工具。

```
liunx@ubuntu:~/Work/Emulator/work$ sudo apt-get install autoconf
liunx@ubuntu:~/Work/Emulator/work$ sudo apt-get install libtool
```

接下来配置编译vde了：

```
liunx@ubuntu:~/Work/Emulator/work$ cd vde_svn; autoreconf -fi; ./configure
--enable-experimental
```

Configure results:

- VDE CryptCab..... disabled
- + VDE Router..... enabled
- Python Libraries..... disabled
- + TAP support..... enabled
- pcap support..... disabled
- + Experimental features... enabled
- Profiling options..... disabled
- Kernel switch..... disabled

configure: WARNING: VDE CryptCab support has been disabled because libcrypto is not installed on your system, or because openssl/blowfish.h could not be found. Please install them if you want CryptCab to be compiled and installed.

configure: WARNING: Python libraries support has been disabled because python is not installed on your system, or because it could not be found. Please install it if you want Python libraries to be compiled and installed.

configure: WARNING: VDE vde\_pcapplug and packet dump plugin have been disabled because libpcap is not installed on your system, or because it is too old. Please install it if you want vde\_pcapplug and pdump to be compiled and installed.

Type 'make' to compile vde2 2.3.2,  
or type 'make V=1' for verbose compiling  
and then type 'make install' to install it into /usr/local

只是有几个功能“disabled”了，说明我们还是缺少相关的开发库的头文件：

```
liunx@ubuntu:~/Work/Emulator/work$ sudo apt-get install libpcap-dev libssl-dev python-all-dev
```

如果一切顺利的话，那么vde相关文件就会安装到/usr/local/这样的好处是不会和系统原装的文件产生冲突，也方便管理了。

# 安装qemu

ubuntu自带的qemu不支持vde选项，所以我们需要自己编译qemu。当然了，编译qemu也需要一些头文件。

```
liunx@ubuntu:~/Work/Emulator/src/qemu$ sudo apt-get install libglib2.0-dev libsdl1.2-dev
libncurses5-dev uuid-dev libcap-ng-dev
liunx@ubuntu:~/Work/Emulator/src/qemu$ ./configure
Install prefix /usr/local
BIOS directory /usr/local/share/qemu
binary directory /usr/local/bin
library directory /usr/local/lib
libexec directory /usr/local/libexec
include directory /usr/local/include
config directory /usr/local/etc
local state directory /usr/local/var
Manual directory /usr/local/share/man
ELF interp prefix /usr/gnemul/qemu-%M
Source path /home/liunx/Work/Emulator/src/qemu
C compiler cc
Host C compiler cc
Objective-C compiler cc
CFLAGS -O2 -D_FORTIFY_SOURCE=2 -g
QEMU_CFLAGS -Werror -fPIE -DPIE -m64 -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64
-D_LARGEFILE_SOURCE -Wstrict-prototypes -Wredundant-decls -Wall -Wundef
-Wwrite-strings -Wmissing-prototypes -fno-strict-aliasing -fstack-protector-all
-Wendif-labels -Wmissing-include-dirs -Wempty-body -Wnested-externs -Wformat-security
-Wformat-y2k -Winit-self -Wignored-qualifiers -Wold-style-declaration -Wold-style-definition
-Wtype-limits -I/usr/include/p11-kit-1 -I/usr/include/libpng12 -I/usr/include/pixman-1
LDFLAGS -Wl,--warn-common -Wl,-z,relro -Wl,-z,now -pie -m64 -g
make make
install install
python python
smbd /usr/sbin/smbd
host CPU x86_64
host big endian no
target list i386-softmmu x86_64-softmmu alpha-softmmu arm-softmmu cris-softmmu
lm32-softmmu m68k-softmmu microblaze-softmmu microblazeel-softmmu mips-softmmu
mipsel-softmmu mips64-softmmu mips64el-softmmu or32-softmmu ppc-softmmu ppcemb-softmmu
ppc64-softmmu sh4-softmmu sh4eb-softmmu sparc-softmmu sparc64-softmmu s390x-softmmu
xtensa-softmmu xtensaeb-softmmu unicore32-softmmu i386-linux-user x86_64-linux-user
alpha-linux-user arm-linux-user arceb-linux-user cris-linux-user m68k-linux-user
microblaze-linux-user microblazeel-linux-user mips-linux-user mipsel-linux-user
or32-linux-user ppc-linux-user ppc64-linux-user ppc64abi32-linux-user sh4-linux-user
sh4eb-linux-user sparc-linux-user sparc64-linux-user sparc32plus-linux-user
unicore32-linux-user s390x-linux-user
tcg debug enabled no
gprof enabled no
sparse enabled no
strip binaries yes
profiler no
static build no
```

-Werror enabled yes  
pixman system  
SDL support yes  
curses support yes  
curl support no  
mingw32 support no  
Audio drivers oss  
Extra audio cards ac97 es1370 sb16 hda  
Block whitelist  
Mixer emulation no  
VirtFS support yes  
VNC support yes  
VNC TLS support yes  
VNC SASL support no  
VNC JPEG support yes  
VNC PNG support yes  
xen support no  
brlapi support no  
bluez support no  
Documentation yes  
NPTL support yes  
GUEST\_BASE yes  
PIE yes  
vde support yes  
Linux AIO support no  
ATTR/XATTR support yes  
Install blobs yes  
KVM support yes  
TCG interpreter no  
fdt support no  
preadv support yes  
fdatasync yes  
madvise yes  
posix\_madvise yes  
sigev\_thread\_id yes  
uuid support yes  
libcap-ng support yes  
vhost-net support yes  
Trace backend nop  
Trace output file trace-<pid>  
spice support no (/)  
rbd support no  
xfstcl support no  
nss used no  
usb net redir no  
OpenGL support yes  
libiscsi support yes  
build guest agent yes  
seccomp support no  
coroutine backend ucontext  
GlusterFS support no  
virtio-blk-data-plane no



这里我们要关注几个配置选项，`kvm enable`，`SDL`和`vde`，这几个配置一定齐全，这样才行，如果不是`yes`，话，那就下载他们的开发库和头文件，`vde`的话如前文所述，已经成功安装了，就不必多虑了。然后`make && make install`，经过漫长的编译（如果不幸你的电脑特别挫的话，其实也不必自责，我的6核大电脑也要忙活一段时间呢），然后自己编译的`qemu`就诞生了！注意安装完成了别忘了看一下`vde`功能是否编译进去了，不然就又要返工了！

如果你的电脑比较古老的话，可能不支持虚拟化技术，那么我们就无法享受到`kvm`带来的性能提升了，不过还好，我们的虚拟机系统很小，负载不会很大的，所以这个影响还不大，只是电脑内存稍微大一些就好。

---

## 第 2 章 搭建网络环境

网络环境之与网络设备就像是水和鱼，谁都离不开谁的，所以我们只是做好了设备，却没有一个可靠的运行环境，那么无论设备再好，也无法发挥他应有的作用。

搭建网络环境也有“内”、“外”之分，“内”指的是主机网络，是指我们的网络环境运行在独立的PC主机上，无需外部设备的参与，优点是显而易见的，就是网络环境本身可以自治，不需要购买其他网络设备，尤其当你处于一个动态IP或者无法上网的情况下，内网环境完全可以由你自己定制，而不受外部的干预，缺点就是单通别的主机想要访问你的内部网络时，主机网络就无能为力了。“外”指的是桥接网络，特点是双通，实现主机之间的互访，但是需要外部的网络设备进行连接，由于受制于外部网络，比如IP分配策略等的限制，使得我们的可定制性下降了。显然，没有任何一种办法是完美的，所以我们要“内外”并用，发挥出他们的功力来。

### 搭建主机网络

搭建主机网络的办法有很多，下面我们就介绍一种方法：

```
liunx@ubuntu:~$ sudo apt-get install virt-manager
```

安装后，我们先运行一下virt-manager，看看他到底是个什么东东：

应用程序菜单 → 系统 → 虚拟系统管理器

不幸的是，virt-manager报错了，“Unable to connect to libvirt”，原来，我们还没有加入libvirtd的用户组，所以没有权限运行，接下来我们需要把当前用户加入到libvirtd的用户组。

应用程序菜单 → 设置管理器 → 用户和组 → 管理组

找到libvirtd组，然后点击属性，打勾当前用户并确定即可，别忘了输入密码。最后注销再登录进去，这样配置就生效了。

接下来开始配置网络了，进入virt-manager主界面，选择：

编辑 → Connection Details

选择“虚拟网络”标签页，这时可以看到有一个default的虚拟网络virbr0，并且自动启动的，为了验证虚拟网络是否启用，我们可以通过终端看到结果：

```
liunx@ubuntu:~/Work/DocBook/Work/code-farmer-note$ ifconfig
eth0  Link encap:以太网 硬件地址 c8:60:00:61:06:16
      inet 地址:192.168.2.100 广播:192.168.2.255 掩码:255.255.255.0
      inet6 地址: fe80::ca60:ff:fe61:616/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1
      接收数据包:5 错误:0 丢弃:0 过载:0 帧数:0
      发送数据包:68 错误:0 丢弃:0 过载:0 载波:1
      碰撞:0 发送队列长度:1000
      接收字节:1360 (1.3 KB) 发送字节:10346 (10.3 KB)
      中断:16

lo    Link encap:本地环回
      inet 地址:127.0.0.1 掩码:255.0.0.0
      inet6 地址: ::1/128 Scope:Host
```

```
UP LOOPBACK RUNNING MTU:16436 跃点数:1
接收数据包:54 错误:0 丢弃:0 过载:0 帧数:0
发送数据包:54 错误:0 丢弃:0 过载:0 载波:0
碰撞:0 发送队列长度:0
接收字节:4327 (4.3 KB) 发送字节:4327 (4.3 KB)
```

```
virbr0 Link encap:以太网 硬件地址 ea:21:c3:5b:e6:1d
inet 地址:192.168.122.1 广播:192.168.122.255 掩码:255.255.255.0
UP BROADCAST MULTICAST MTU:1500 跃点数:1
接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
发送数据包:0 错误:0 丢弃:0 过载:0 载波:0
碰撞:0 发送队列长度:0
接收字节:0 (0.0 B) 发送字节:0 (0.0 B)
```

看，virbr0就是我们所需要的主机网络！当然了，你还可以添加更多的虚拟网络，不过默认目前已经够用了。

## 搭建桥接网络

桥接网络我们同样可以通过virt-manager添加，但是virt-manager默认并没有创建桥接网络的，所以需要我们动手设置。打开虚拟系统管理器，进入Connection Details选择网络接口，点击添加按钮，这是出现一个设置对话框，Interface type选择桥接，然后点击下一步，名称使用默认的br0即可，当然你也可以选择一个拉风的名字，Start mode选择onboot，Active now勾选上吧，这样保证配置立即生效，IP settings可以选择DHCP，也可以选择Static，最后选择桥接绑定的物理接口，这里选择eth0，当然了，如果你有多块网卡的话，也可以选择别的接口，但是不要选择lo，这是系统内部的回路系统，显然是不行的！最后点击完成，然后去上个厕所什么的，因为virt-manager创建桥接的时候会耗费一些时间。

最终我们得到的就是下面的结果：

```
liunix@ubuntu:~/Work/DocBook/Work/code-farmer-note$ ifconfig
br0 Link encap:以太网 硬件地址 c8:60:00:61:06:16
inet 地址:192.168.2.100 广播:255.255.255.255 掩码:255.255.255.0
inet6 地址: fe80::ca60:ff:fe61:616/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1
接收数据包:767 错误:0 丢弃:0 过载:0 帧数:0
发送数据包:8037 错误:0 丢弃:0 过载:0 载波:0
碰撞:0 发送队列长度:0
接收字节:37170 (37.1 KB) 发送字节:603782 (603.7 KB)

eth0 Link encap:以太网 硬件地址 c8:60:00:61:06:16
UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1
接收数据包:11267 错误:0 丢弃:0 过载:0 帧数:0
发送数据包:8037 错误:0 丢弃:0 过载:0 载波:1
碰撞:0 发送队列长度:1000
接收字节:3800860 (3.8 MB) 发送字节:616766 (616.7 KB)
中断:16

lo Link encap:本地环回
inet 地址:127.0.0.1 掩码:255.0.0.0
inet6 地址: ::1/128 Scope:Host
```

```
UP LOOPBACK RUNNING MTU:16436 跃点数:1
接收数据包:24 错误:0 丢弃:0 过载:0 帧数:0
发送数据包:24 错误:0 丢弃:0 过载:0 载波:0
碰撞:0 发送队列长度:0
接收字节:2301 (2.3 KB) 发送字节:2301 (2.3 KB)
```

```
virbr0 Link encap:以太网 硬件地址 d6:0f:1c:fd:6c:64
inet 地址:192.168.122.1 广播:192.168.122.255 掩码:255.255.255.0
UP BROADCAST MULTICAST MTU:1500 跃点数:1
接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
发送数据包:0 错误:0 丢弃:0 过载:0 载波:0
碰撞:0 发送队列长度:0
接收字节:0 (0.0 B) 发送字节:0 (0.0 B)
```

## 搭建VDE网络

VDE的全称叫做虚拟分布式网络说得通俗一点就是虚拟网络设备，当然VDE项目的目标远远不止如此，但是对于我们来说，这样理解已经够用了。再用简单的语言解释一下，VDE就是通过程序来模拟网络设备，比如交换机（vde\_switch），路由器（slirpvde）等。好，接下来让我们看看他的庐山真面目吧：

```
liunx@ubuntu:~/Work/Emulator/work/vde$ vde_switch -s /tmp/switch
```

```
vde$
```

嗯，看样子没什么反应啊，这是什么情况？好吧，接下来就让我对这个命令做个解释，你就会明白了。vde\_switch用来模拟交换机设备的，有了交换机，我们就可以接设备了，但是总要有个接口才行啊，好那么我们继续往下看：

```
liunx@ubuntu:~$ ls /tmp/switch/
ctl
liunx@ubuntu:~$
```

这里vde\_switch创建了名为/tmp/switch/ctl的socket文件，这里就是链接虚拟交换机的接口，我们只要在qemu中指定这个文件，就可以连接到虚拟交换机上了，下面看看我们的配置：

```
liunx@ubuntu:~/Work/OpenWrt/work/x86$ cat boot.sh
#!/bin/bash -
set -o nounset                # Treat unset variables as an error

# -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:01 \
# -net vde,sock=/tmp/vde.ctl
# -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:01 \
# -net tap,vlan=0,ifname=openwrt0,script=qemu-ifup,downscript=qemu-ifdown \

/usr/local/bin/qemu-system-i386 \
-m 64M \
-kernel openwrt-x86-generic-vmlinuz \
```

```
-hda openwrt-x86-generic-rootfs-ext4.img \  
-append "root=/dev/sda console=ttyS0" \  
-enable-kvm \  
-nographic \  
-net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:01 \  
-net vde,sock=/tmp/switch
```

看我们最后的-net配置，上面的配置是制定网卡模型和MAC地址，这个是通用的，主要是后面的配置我们指定了vde网络，这样我们的VDE网络就配置好了。

## slirpvde

slirpvde是一个用户空间的虚拟路由器，我们可以把slirpvde加入到交换机网络中来动态分配地址，同时通过slirpvde，交换机网络中的其他主机可以访问外部网络，但是遗憾的是，slirpvde却无法对ICMP测试做地址转换，也就是说当我们ping外部网络时，你会发现无响应的。不过，没关系，我们可以正常访问外部网络了，而无需超级用户权限了。

所以我们的vde switch网络可以这样设置：

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images/custom$ cat switch.sh  
#!/bin/bash -  
set -o nounset                # Treat unset variables as an error  
  
echo "begin switch..."  
vde_switch --daemon --sock /tmp/switch --mgmt /tmp/switch.mgmt  
slirpvde -d -s /tmp/switch -dhcp  
echo "OK"
```

## 第 3 章 搭建嵌入式系统

前面我们准备好了开发环境，现在是演职人员登场的时候了。这里是基于linux的嵌入式系统。

### 运行OpenWrt

首先登场的是OpenWrt，这是一款很出名的开源路由器系统，当然是基于linux的了，那么OpenWrt已经就是一个路由系统了，作为嵌入式系统，他的体积很小，启动速度很快，加上其支持X86平台，很符合我们的要求。而且他包含了大量的开源软件，使得我们不必满世界的移植程序了，Perfect！

获得OpenWrt系统的方式有以下几种：

1. 从源码进行编译
2. 获取编译好的二进制镜像系统

编译OpenWrt也不难，可是时间上实在是太长了，如果你的电脑不够Power的话，那么还是省省吧，还好，OpenWrt为我们编译好了各种平台上镜像版本，同时还包括软件仓库，这样既节约了时间，又方便新手入门，只是一举多得。

好，那么去哪下载官方的二进制镜像呢，看这里<http://downloads.openwrt.org/>。



| Index of /                           |                   |   |
|--------------------------------------|-------------------|---|
| <a href="#">../</a>                  |                   |   |
| <a href="#">attitude_adjustment/</a> | 27-Nov-2012 22:53 | - |
| <a href="#">backfire/</a>            | 28-Sep-2012 19:50 | - |
| <a href="#">backports/</a>           | 28-Sep-2012 19:53 | - |
| <a href="#">docs/</a>                | 08-Nov-2011 12:29 | - |
| <a href="#">kamikaze/</a>            | 28-Sep-2012 19:16 | - |
| <a href="#">people/</a>              | 09-Jun-2009 09:53 | - |
| <a href="#">reference/</a>           | 06-Aug-2005 01:10 | - |
| <a href="#">snapshots/</a>           | 23-Oct-2012 14:51 | - |
| <a href="#">sources/</a>             | 03-Jan-2013 13:49 | - |
| <a href="#">utils/</a>               | 06-Apr-2008 15:50 | - |
| <a href="#">whiterussian/</a>        | 28-Sep-2012 19:13 | - |
| <a href="#">favicon.ico</a>          | 02-Jul-2005 04:41 | 0 |

这么多选择，该选哪一个呢？这里我们关注有以下几个版本入口：

1. backfire
2. kamikaze
3. snapshots
4. attitude\_adjustment

backfire和kamikaze属于两个稳定的分支，正常使用没有什么问题，就是版本比较老了，相应的软件包也比较老，attitude\_adjustment则是最近的稳定版本，这里我们可以使用他们的x86版本，但是这里我们选择snapshots版本，虽然快照版本相对来说不是很稳定，但是代码更新很快，内核版本，软件包也更新，其实快照版本就是用来给大家测试用的，以便获取错误信息的反馈，增强系统的稳定性，所以，本着参与的目的，我们选择snapshots版本，如果有

问题的话，我们还可以反馈一些内容了。但是，需要注意的是，由于这是个快照版本，内核模块重新编译的话，是使得安装内核模块时失败，还好整个系统不是很大，我们只要及时跟新镜像就是了，而且也用不到所有的软件包的，所以够用了。

这里我们选择attitude\_adjustment版本吧，可以避免一些不必要的麻烦。

接下来，我们进入attitude\_adjustment目录，下载镜像。

1. [http://downloads.openwrt.org/attitude\\_adjustment/12.09-rc1/x86/generic/openwrt-x86-generic-rootfs-ext4.img.gz](http://downloads.openwrt.org/attitude_adjustment/12.09-rc1/x86/generic/openwrt-x86-generic-rootfs-ext4.img.gz)

2. [http://downloads.openwrt.org/attitude\\_adjustment/12.09-rc1/x86/generic/openwrt-x86-generic-vmlinuz](http://downloads.openwrt.org/attitude_adjustment/12.09-rc1/x86/generic/openwrt-x86-generic-vmlinuz)

3. [http://downloads.openwrt.org/attitude\\_adjustment/12.09-rc1/x86/generic/openwrt-x86-generic-combined-ext4.img.gz](http://downloads.openwrt.org/attitude_adjustment/12.09-rc1/x86/generic/openwrt-x86-generic-combined-ext4.img.gz)

openwrt-x86-generic-combined-ext4.img.gz包括了引导程序、内核和跟文件系统，优点是简单易用，缺点是无法挂载，所以导入配置文件的时候有些麻烦了，但是也没关系的，如果我们安装了openWrt自带的用户界面的话，可以导入/导出配置文件的，但是目前来说作用还不是很大，反而增加了复杂性，所以我们选择后面两个镜像来配合使用，前者只包含了根文件系统，后者是内核。

好了，别忘了解压zip文件。下面，见证奇迹的时刻到了，前面做了那么多艰苦卓绝的工作，该是出成果的时候了！先看一下我们的文件布局：

```
liunix@liunix-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ tree
```

```
.
├── boot.sh
├── images
│   ├── openwrt-x86-generic-combined-ext4.img
│   ├── openwrt-x86-generic-rootfs-ext4.img
│   └── openwrt-x86-generic-vmlinuz
├── qemu-ifdown
└── qemu-ifup
```

2 directories, 7 files

再看我们的运行脚本：

```
liunix@liunix-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ ls
boot.sh images qemu-ifdown qemu-ifup
liunix@liunix-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ cat qemu-ifdown
#!/bin/bash -

bridge=virbr0
brctl delif $bridge $1 || true
ifconfig $1 down
liunix@liunix-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ cat qemu-ifup
#!/bin/bash -
```

```
bridge=virbr0
ifconfig $1 0.0.0.0 promisc up
brctl addif $bridge $1
liunx@liunx-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ cat boot.sh
#!/bin/bash -

set -o nounset                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel images/openwrt-x86-generic-vmlinuz \
    -hda images/openwrt-x86-generic-rootfs-ext4.img \
    -append "root=/dev/sda console=ttyS0" \
    -enable-kvm \
    -nographic \
    -usb \
    -net nic,vlan=3,model=e1000,macaddr="12:34:56:78:9a:00" \
    -net tap,vlan=3,ifname=openwrt0,script=qemu-ifup,downscript=qemu-ifdown
```

看好了，大家！这里我们需要五个文件：

- 1.boot.sh
- 2.qemu-ifup
- 3.qemu-ifdown
- 4.images/openwrt-x86-generic-rootfs-ext4.img
- 5.images/openwrt-x86-generic-vmlinuz

你都找到了吗？

等等，别忘了给脚本加上可执行权限！

```
liunx@liunx-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ chmod +x boot.sh qemu-ifup qemu-ifdown
```

最后，激动人心的时刻到了，运行！

```
liunx@liunx-G41MT-S2PT:~/Work/OpenWrt/work/openwrt$ sudo ./boot.sh
```





Build:

all - make world  
 <package>-rebuild - force recompile <package>  
 <package>-reconfigure - force reconfigure <package>

Configuration:

menuconfig - interactive curses-based configurator  
 nconfig - interactive ncurses-based configurator  
 xconfig - interactive Qt-based configurator  
 gconfig - interactive GTK-based configurator  
 oldconfig - resolve any unresolved symbols in .config  
 randconfig - New config with random answer to all options  
 defconfig - New config with default answer to all options  
 BR2\_DEFCONFIG, if set, is used as input  
 savedefconfig - Save current config as ./defconfig (minimal config)  
 allyesconfig - New config where all options are accepted with yes  
 allnoconfig - New config where all options are answered with no  
 randpackageconfig - New config with random answer to package options  
 allyespackageconfig - New config where pkg options are accepted with yes  
 allnopackageconfig - New config where package options are answered with no

Documentation:

manual - build manual in HTML, split HTML, PDF and txt  
 manual-html - build manual in HTML  
 manual-split-html - build manual in split HTML  
 manual-pdf - build manual in PDF  
 manual-txt - build manual in txt  
 manual-epub - build manual in ePub

Miscellaneous:

source - download all sources needed for offline-build  
 source-check - check selected packages for valid download URLs  
 external-deps - list external packages used  
 legal-info - generate info about license compliance  
  
 make V=0|1 - 0 => quiet build (default), 1 => verbose build  
 make O=dir - Locate all output files in "dir", including .config

arm\_foundationv8\_defconfig - Build for arm\_foundationv8  
 armadeus\_apf9328\_defconfig - Build for armadeus\_apf9328  
 at91rm9200df\_defconfig - Build for at91rm9200df  
 at91sam9260dfc\_defconfig - Build for at91sam9260dfc  
 at91sam9261ek\_defconfig - Build for at91sam9261ek  
 at91sam9263ek\_defconfig - Build for at91sam9263ek  
 at91sam9g20dfc\_defconfig - Build for at91sam9g20dfc  
 atngw100\_defconfig - Build for atngw100  
 atstk100x\_defconfig - Build for atstk100x  
 beaglebone\_defconfig - Build for beaglebone  
 calao\_qil\_a9260\_defconfig - Build for calao\_qil\_a9260  
 calao\_usb\_a9263\_defconfig - Build for calao\_usb\_a9263  
 calao\_usb\_a9g20\_lpw\_defconfig - Build for calao\_usb\_a9g20\_lpw  
 ea3250\_defconfig - Build for ea3250  
 fdi3250\_defconfig - Build for fdi3250

|                                 |                                   |
|---------------------------------|-----------------------------------|
| integrator926_defconfig         | - Build for integrator926         |
| kb9202_defconfig                | - Build for kb9202                |
| mini2440_defconfig              | - Build for mini2440              |
| mx53loco_defconfig              | - Build for mx53loco              |
| nitrogen6x_defconfig            | - Build for nitrogen6x            |
| pandaboard_defconfig            | - Build for pandaboard            |
| phy3250_defconfig               | - Build for phy3250               |
| qemu_arm_versatile_defconfig    | - Build for qemu_arm_versatile    |
| qemu_arm_vexpress_defconfig     | - Build for qemu_arm_vexpress     |
| qemu_microblazebe_mmu_defconfig | - Build for qemu_microblazebe_mmu |
| qemu_microblazeel_mmu_defconfig | - Build for qemu_microblazeel_mmu |
| qemu_mips64_malta_defconfig     | - Build for qemu_mips64_malta     |
| qemu_mips_malta_defconfig       | - Build for qemu_mips_malta       |
| qemu_mipsel_malta_defconfig     | - Build for qemu_mipsel_malta     |
| qemu_ppc_g3beige_defconfig      | - Build for qemu_ppc_g3beige      |
| qemu_ppc_mpc8544ds_defconfig    | - Build for qemu_ppc_mpc8544ds    |
| qemu_sh4_r2d_defconfig          | - Build for qemu_sh4_r2d          |
| qemu_sparc_ss10_defconfig       | - Build for qemu_sparc_ss10       |
| qemu_x86_64_defconfig           | - Build for qemu_x86_64           |
| qemu_x86_defconfig              | - Build for qemu_x86              |
| s6lx9_microboard_defconfig      | - Build for s6lx9_microboard      |
| sheevaplug_defconfig            | - Build for sheevaplug            |

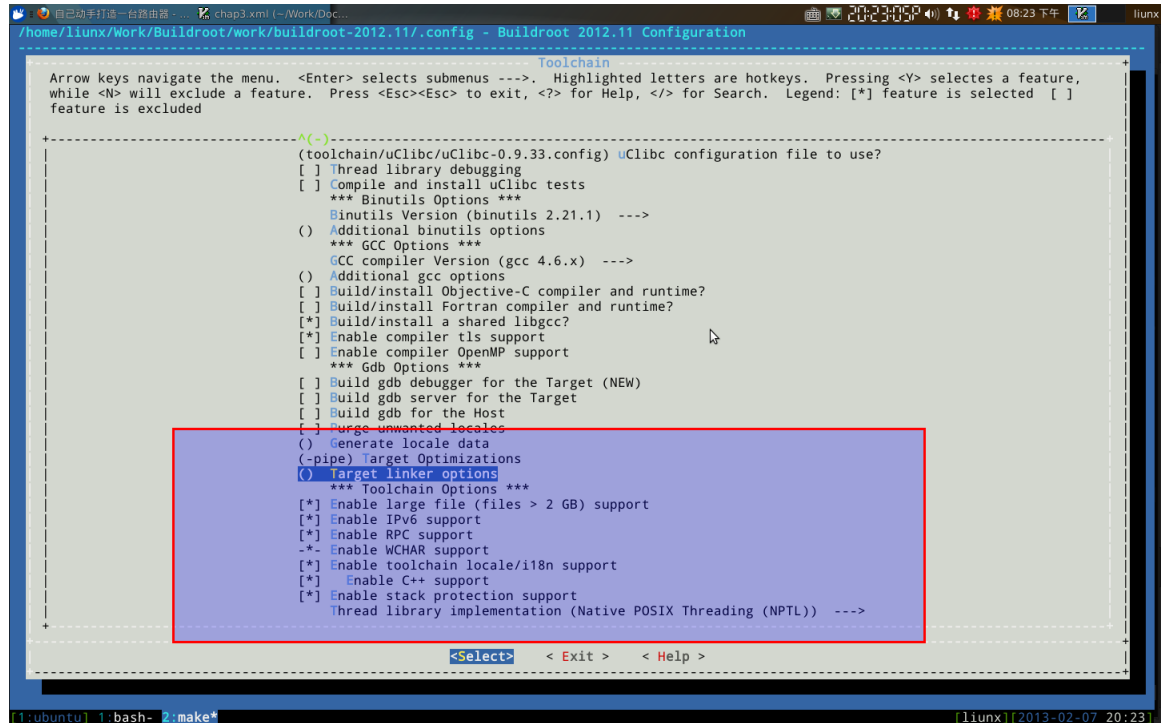
See docs/README, or generate the Buildroot manual for further details

看qemu\_x86\_defconfig，我们要的就是他：

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11$ make qemu_x86_defconfig
mkdir -p /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/lxdialog
make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" obj=/home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config -C support/kconfig -f Makefile.br conf
make[1]: 正在进入目录 `/home/liunx/Work/Buildroot/work/buildroot-2012.11/support/kconfig'
/usr/bin/gcc -DCURSES_LOC="" -DLOCALE -I/home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config -MM *.c > /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/.depend 2>/dev/null || :
make[1]:正在离开目录 `/home/liunx/Work/Buildroot/work/buildroot-2012.11/support/kconfig'
make[1]: 正在进入目录 `/home/liunx/Work/Buildroot/work/buildroot-2012.11/support/kconfig'
/usr/bin/gcc -DCURSES_LOC="" -DLOCALE -I/home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config -c conf.c -o /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/conf.o
/usr/bin/gcc -DCURSES_LOC="" -DLOCALE -I/home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config -I. -c /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/zconf.tab.c -o /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/zconf.tab.o
/usr/bin/gcc -DCURSES_LOC="" -DLOCALE -I/home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/conf.o /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/zconf.tab.o -o /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/conf
rm /home/liunx/Work/Buildroot/work/buildroot-2012.11/output/build/buildroot-config/zconf.tab.c
make[1]:正在离开目录 `/home/liunx/Work/Buildroot/work/buildroot-2012.11/support/kconfig'
#
```

```
# configuration written to /home/liunx/Work/Buildroot/work/buildroot-2012.11/.config
#
```

生成默认配置了，接下来我们要做一些定制了。首先定制一下toolchain，默认的工具链支持的功能较少，比如不支持C++，所以为了以后能编译更多的软件，我们尽量打开所有的功能，因为buildroot会自动编译工具链的，而这个编译又很费时间，所以我们最好是一次成型。



注意红色区域，这里是重点，当然一些杂七杂八的东西你也可以勾选，但要记住，“宁可错杀一千，不可放过一个”！接下来定制软件包，默认配置的软件包很少，这里我们要添加一些网络相关的软件包。进入Package Selection for the target，然后勾选Show package that are also provided by busbox，因为busybox的替代功能有限，还是用原装的好。进入Networking applications这里，我们根据需要选择一些应用程序即可。

还有一个细节的地方需要修改，我们需要系统将信息输出到串口，这样方便使用，所以我们将System configuration → Port to run a getty (login prompt) on 的tty1改为ttyS0，这样我们可以充分利用qemu的-nographic选项，直接在终端里显示，而不产生麻烦的窗口界面了，毕竟我们的设备没有图像界面，使用窗口显示真是很麻烦的。

最后，保存配置，退出，然后make，耐心等待吧，如果网络顺畅，电脑给力的话，一个小时内应该可以解决战斗的。buildroot会自动根据你的电脑CPU核心数来分配线程的，充分利用资源，其实花时间的地方源于下载代码，这里要看你的网络情况了。

编译完成后，生成的镜像在output/images目录。

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11$ ls output/images/
bzImage rootfs.ext2
```

---

## 第 4 章 设备集成

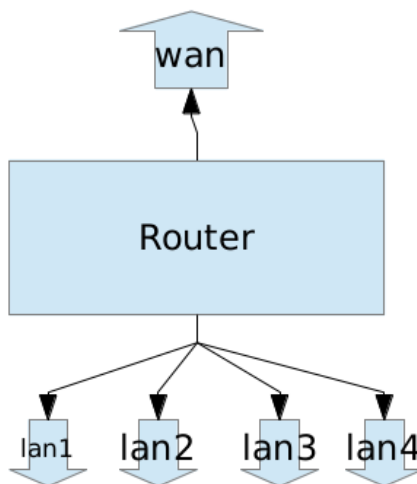
前面，我们介绍了如何搭建运行环境，网络环境，还有运行嵌入式系统，但是一切的一切也只是万里长征第一步，接下来我们就要到设计和集成的阶段了。

### 组装路由器

一台路由器究竟由什么构成的呢？当然了，我们不是在讨论硬件电路的设计问题，不用关心电源了，电路布局之类的事情，我们关心是一台路由器是由哪些功能部分组成的。

首先，路由器的职责就是帮助我们上网的，那是肯定的了，要上网，那至少要有个WAN口了，WAN口，顾名思义，就是连接到电信运营商链路的端口了，他的前端可能是一只猫，两只狗，三只羊什么的东东，好了，那些都是运营商的工作，我们就不必关心了。

好，接下来开始动手了！首先看看我们的路由器应该是这幅模样：



这里，我们设计的路由器功能是有1个wan口，4个lan口组成。看起来像一只章鱼，我们暂且给他起个名字叫“老章”吧，“老章”设计草图出来了，下面进入建造阶段：

下面看看我们的qemu的配置：

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ ls
boot.sh bzImage rootfs.ext2
```

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ cat boot.sh
#!/bin/bash -
set -o nounset                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel images/bzImage \
    -hda images/rootfs.ext2 \
    -append "root=/dev/sda console=ttyS0" \
```

```

-enable-kvm \
-nographic \
-net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:01 \
-net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:12 \
-net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:13 \
-net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:14 \
-net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:15

```

从配置参数中可以看出，我们创建了5个网卡，基本上，“老章”的躯体就算是出来了。可是奇怪的是，根本无法区别谁是wan口，谁是lan口啊。接下来需要系统来划分功能了。默认linux配置并不支持e1000的网卡，所以我们需要配置内核：进入buildroot顶层目录，然后make linux-menuconfig 此时我们进入了内核配置界面，勾选 Device Drivers → Network device support → Ethernet driver support → Intel(R) PRO/1000 Gigabit Ethernet support 此时，内核就可以识别和驱动qemu e1000的网卡模型了。好，保存退出，然后重新 make。

OK，现在让我运行一下看看效果：

```

liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ ./boot.sh
Warning: vlan 0 is not connected to host network
Linux version 3.6.6 (liunx@ubuntu) (gcc version 4.6.3 (Buildroot 2012.11)) #2 Thu Feb 7 22:59:51 CST 2013
e820: BIOS-provided physical RAM map:
BIOS-e820: [mem 0x0000000000000000-0x0000000000009fbfff] usable
BIOS-e820: [mem 0x0000000000009fc00-0x0000000000009ffff] reserved
BIOS-e820: [mem 0x000000000000f0000-0x000000000000ffffff] reserved
BIOS-e820: [mem 0x00000000000100000-0x000000000003ffdf] usable
BIOS-e820: [mem 0x000000000003ffe000-0x000000000003ffffff] reserved
BIOS-e820: [mem 0x000000000feffc000-0x000000000feffffff] reserved
BIOS-e820: [mem 0x00000000fffc0000-0x00000000ffffff] reserved
Notice: NX (Execute Disable) protection missing in CPU!
DMI 2.4 present.
e820: last_pfn = 0x3ffe max_arch_pfn = 0x100000
init_memory_mapping: [mem 0x00000000-0x03ffdf]
ACPI: RSDP 000fd8c0 00014 (v00 BOCHS )
ACPI: RSDT 03ffe4b0 00034 (v01 BOCHS BXP CRSDT 00000001 BXP 00000001)
ACPI: FACP 03ffff80 00074 (v01 BOCHS BXP CFACP 00000001 BXP 00000001)
ACPI: DSDT 03ffe4f0 011A9 (v01 BXP BXDSDT 00000001 INTL 20090123)
ACPI: FACS 03ffff40 00040
ACPI: SSDT 03fff800 00735 (v01 BOCHS BXP CSSDT 00000001 BXP 00000001)
ACPI: APIC 03fff6e0 00078 (v01 BOCHS BXP APIC 00000001 BXP 00000001)
ACPI: HPET 03fff6a0 00038 (v01 BOCHS BXP HPET 00000001 BXP 00000001)
0MB HIGHMEM available.
63MB LOWMEM available.
 mapped low ram: 0 - 03ffe000
 low ram: 0 - 03ffe000
Zone ranges:
 DMA [mem 0x00010000-0x00ffffff]
 Normal [mem 0x01000000-0x03ffdf]
 HighMem empty
Movable zone start for each node
Early memory node ranges
 node 0: [mem 0x00010000-0x0009eff]

```

```

node 0: [mem 0x00100000-0x03ffdfbf]
ACPI: PM-Timer IO Port: 0xb008
e820: [mem 0x04000000-0xfffbfff] available for PCI devices
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16141
Kernel command line: root=/dev/sda console=ttyS0
PID hash table entries: 256 (order: -2, 1024 bytes)
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
__ex_table already sorted, skipping sort
Initializing CPU#0
Initializing HighMem for node 0 (00000000:00000000)
Memory: 60376k/65528k available (2611k kernel code, 4700k reserved, 903k data, 284k init, 0k highmem)
virtual kernel memory layout:
  fixmap : 0xffff4000 - 0xffff0000 ( 108 kB)
  pkmap : 0xff800000 - 0xffc00000 (4096 kB)
  vmalloc : 0xc47fe000 - 0xff7fe000 ( 944 MB)
  lowmem : 0xc0000000 - 0xc3ffe000 ( 63 MB)
  .init : 0xc136f000 - 0xc13b6000 ( 284 kB)
  .data : 0xc128cf3b - 0xc136ec60 ( 903 kB)
  .text : 0xc1000000 - 0xc128cf3b (2611 kB)
Checking if this processor honours the WP bit even in supervisor mode...Ok.
SLUB: Genslabs=15, HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
NR_IRQS:16 nr_irqs:16 16
Console: colour VGA+ 80x25
console [ttyS0] enabled
tsc: Fast TSC calibration using PIT
tsc: Detected 3013.507 MHz processor
Calibrating delay loop (skipped), value calculated using timer frequency.. 6027.01 BogoMIPS (lpj=12054028)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
Last level iTLB entries: 4KB 0, 2MB 0, 4MB 0
Last level dTLB entries: 4KB 0, 2MB 0, 4MB 0
tlb_flushall_shift is 0xffffffff
CPU: AMD QEMU Virtual CPU version 1.3.50 stepping 03
ACPI: Core revision 20120711
ACPI: setting ELCR to 0200 (from 0c00)
Performance Events: Broken PMU hardware detected, using software events only.
Failed to access perfctr msr (MSR c0010001 is ffffffffffffffff)
NET: Registered protocol family 16
ACPI: bus type pci registered
PCI: PCI BIOS revision 2.10 entry at 0xffe77, last bus=0
PCI: Using configuration type 1 for base access
bio: create slab <bio-0> at 0
ACPI: Added _OSI(Module Device)
ACPI: Added _OSI(Processor Device)
ACPI: Added _OSI(3.0 _SCP Extensions)
ACPI: Added _OSI(Processor Aggregator Device)
ACPI: Interpreter enabled
ACPI: (supports S0 S3 S5)
ACPI: Using PIC for interrupt routing
PCI: Using host bridge windows from ACPI; if necessary, use "pci=nocrs" and report a bug
ACPI: PCI Root Bridge [PCI0] (domain 0000 [bus 00-ff])
pci_root PNP0A03:00: fail to add MMCONFIG information, can't access extended PCI configuration
space under this bridge.

```

```

PCI host bridge to bus 0000:00
pci_bus 0000:00: root bus resource [bus 00-ff]
pci_bus 0000:00: root bus resource [io 0x0000-0x0cf7]
pci_bus 0000:00: root bus resource [io 0x0d00-0xffff]
pci_bus 0000:00: root bus resource [mem 0x000a0000-0x000bffff]
pci_bus 0000:00: root bus resource [mem 0x80000000-0xfebfffff]
pci 0000:00:01.3: quirk: [io 0xb000-0xb03f] claimed by PIIX4 ACPI
pci 0000:00:01.3: quirk: [io 0xb100-0xb10f] claimed by PIIX4 SMB
pci0000:00: Unable to request _OSC control (_OSC support mask: 0x08)
ACPI: PCI Interrupt Link [LNKA] (IRQs 5 *10 11)
ACPI: PCI Interrupt Link [LNKB] (IRQs 5 *10 11)
ACPI: PCI Interrupt Link [LNKC] (IRQs 5 10 *11)
ACPI: PCI Interrupt Link [LNKD] (IRQs 5 10 *11)
ACPI: PCI Interrupt Link [LNKS] (IRQs *9)
vgaarb: device added: PCI:0000:00:02.0,decodes=io+mem,owns=io+mem,locks=None
vgaarb: loaded
vgaarb: bridge control possible 0000:00:02.0
SCSI subsystem initialized
ACPI: bus type scsi registered
Advanced Linux Sound Architecture Driver Version 1.0.25.
PCI: Using ACPI for IRQ routing
Switching to clocksource pit
pnp: PnP ACPI init
ACPI: bus type pnp registered
pnp: PnP ACPI: found 8 devices
ACPI: ACPI bus type pnp unregistered
Switching to clocksource acpi_pm
NET: Registered protocol family 2
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP: reno registered
UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
pci 0000:00:00.0: Limiting direct PCI/PCI transfers
pci 0000:00:01.0: PIIX3: Enabling Passive Release
pci 0000:00:01.0: Activating ISA DMA hang workarounds
io scheduler noop registered (default)
input: Power Button as /devices/LNXSYSTM:00/LNXPWRBN:00/input/input0
ACPI: Power Button [PWRB]
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
00:06: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
scsi0 : ata_piix
scsi1 : ata_piix
ata1: PATA max MWDMA2 cmd 0x1f0 ctl 0x3f6 bmdma 0xc140 irq 14
ata2: PATA max MWDMA2 cmd 0x170 ctl 0x376 bmdma 0xc148 irq 15

e1000: Intel(R) PRO/1000 Network Driver - version 7.3.21-k8-NAPI

```



e1000: Copyright (c) 1999-2006 Intel Corporation.

```
ACPI: PCI Interrupt Link [LNKC] enabled at IRQ 11
ata2.00: ATAPI: QEMU DVD-ROM, 1.3.50, max UDMA/100
ata1.00: ATA-7: QEMU HARDDISK, 1.3.50, max UDMA/100
ata1.00: 16494 sectors, multi 16: LBA48
ata2.00: configured for MWDMA2
ata1.00: configured for MWDMA2
scsi 0:0:0:0: Direct-Access  ATA  QEMU HARDDISK  1.3. PQ: 0 ANSI: 5
scsi 1:0:0:0: CD-ROM      QEMU  QEMU DVD-ROM   1.3. PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 16494 512-byte logical blocks: (8.44 MB/8.05 MiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
sda: unknown partition table
sd 0:0:0:0: [sda] Attached SCSI disk
e1000 0000:00:03:0: eth0: (PCI:33MHz:32-bit) 52:54:00:12:34:01
e1000 0000:00:03:0: eth0: Intel(R) PRO/1000 Network Connection
ACPI: PCI Interrupt Link [LNKD] enabled at IRQ 11
e1000 0000:00:04:0: eth1: (PCI:33MHz:32-bit) 52:54:00:12:34:12
e1000 0000:00:04:0: eth1: Intel(R) PRO/1000 Network Connection
ACPI: PCI Interrupt Link [LNKA] enabled at IRQ 10
tsc: Refined TSC clocksource calibration: 3013.705 MHz
Switching to clocksource tsc
e1000 0000:00:05:0: eth2: (PCI:33MHz:32-bit) 52:54:00:12:34:13
e1000 0000:00:05:0: eth2: Intel(R) PRO/1000 Network Connection
ACPI: PCI Interrupt Link [LNKB] enabled at IRQ 10
e1000 0000:00:06:0: eth3: (PCI:33MHz:32-bit) 52:54:00:12:34:14
e1000 0000:00:06:0: eth3: Intel(R) PRO/1000 Network Connection
e1000 0000:00:07:0: eth4: (PCI:33MHz:32-bit) 52:54:00:12:34:15
e1000 0000:00:07:0: eth4: Intel(R) PRO/1000 Network Connection
i8042: PNP: PS/2 Controller [PNP0303:KBD,PNP0f13:MOU] at 0x60,0x64 irq 1,12
serio: i8042 KBD port at 0x60,0x64 irq 1
serio: i8042 AUX port at 0x60,0x64 irq 12
mousedev: PS/2 mouse device common for all mice
cpuidle: using governor ladder
TCP: cubic registered
NET: Registered protocol family 17
input: AT Translated Set 2 keyboard as /devices/platform/i8042/serio0/input/input1
ALSA device list:
  No soundcards found.
VFS: Mounted root (ext2 filesystem) readonly on device 8:0.
Freeing unused kernel memory: 284k freed
EXT2-fs (sda): warning: mounting unchecked fs, running e2fsck is recommended
Starting logging: OK
Initializing random number generator... done.
Starting network...
udhcpd (v1.20.2) started
Sending discover...
e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Sending discover...
Sending discover...
No lease, failing
```

Welcome to Buildroot

```
buildroot login: root
#
# ifconfig -a
eth0   Link encap:Ethernet HWaddr 52:54:00:12:34:01
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:0 errors:0 dropped:0 overruns:0 frame:0
       TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:0 (0.0 B)  TX bytes:644 (644.0 B)

eth1   Link encap:Ethernet HWaddr 52:54:00:12:34:12
       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)

eth2   Link encap:Ethernet HWaddr 52:54:00:12:34:13
       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)

eth3   Link encap:Ethernet HWaddr 52:54:00:12:34:14
       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)

eth4   Link encap:Ethernet HWaddr 52:54:00:12:34:15
       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)

lo     Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       UP LOOPBACK RUNNING  MTU:16436  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:0
       RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

这里我们没有创建tap连接，因此不需要超级用户权限。buildroot提示登录账户时，我们只需输入“root”即可，不需要密码。运行ifconfig -a可以看到，5块网卡赫然在目。

问： 妈呀，qemu退不出来了！坑爹啊这是？！

答： 大哥，想退出您就poweroff，再不行您就Ctrl-a加x 保管行！

## lan侧桥接设置

其实，路由器的wan/lan的区分是由软件系统进行功能划分的，lan侧和wan侧的功能显然是有区别的，wan侧网络只需获取一个有效的地址即可工作了，而lan侧则需要桥接进行统一管理。

### 选材

lan侧桥接的设备需要两个层面的配置，一个是内核打开桥接功能，另一个是应用层软件进行桥接设置。首先，配置内核：make linux-menuconfig，勾选 Networking support → Networking options → 802.1d Ethernet Bridging，保存退出。接下来 make menuconfig，进入 Package Selection for the target → Networking applications勾选“Bridge-utils”，然后勾选“dnsmasq”，这时会出现多个选项，我们只需勾选“dhcp support”就足够了，最后保存退出，make！

好，编译成功并不意味着lan侧桥接就搞定了，我们还要添加相关的启动脚本和配置文件。

### 添加vde连接

上文中我们说到组装“老章”的躯体，虽然网卡装上了，但是却没有和外界联系的接口，显然是个“中看不中用”的货色，这怎么行呢？那么，我们需要给他提供一个和外部连接的线了：

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ cat boot.sh
#!/bin/bash -
set -o nounset                                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel bzImage \
    -hda rootfs.ext2 \
    -append "root=/dev/sda console=ttyS0" \
    -enable-kvm \
    -nographic \
    -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:01 \
    -net nic,vlan=1,model=e1000,macaddr=52:54:00:12:34:11 \
    -net vde,vlan=1,sock=/tmp/lan1ctl[] \
    -net nic,vlan=2,model=e1000,macaddr=52:54:00:12:34:12 \
    -net vde,vlan=2,sock=/tmp/lan2ctl[] \
    -net nic,vlan=3,model=e1000,macaddr=52:54:00:12:34:13 \
    -net vde,vlan=3,sock=/tmp/lan3ctl[] \
    -net nic,vlan=4,model=e1000,macaddr=52:54:00:12:34:14 \
    -net vde,vlan=4,sock=/tmp/lan4ctl[]
```

对比上文中的boot.sh，我们首先为每个lan口配置了独立的vlan ID，同时每个lan口会创建一个socket文件来作为连接的“线”，OK，这样，“老章”终于可以干点活了。

### 添加bridge脚本

我们的嵌入式系统显然不会自动创建桥接接口的，所以需要我们手动添加了，那么看脚本吧：

```
# cat /usr/lib/scripts/bridge
```

```
#!/bin/sh

BRNAME=br0
BINDIFS="eth1 eth2 eth3 eth4"

bridge_start() {
    echo "bridge_start..."
    brctl addbr $BRNAME
    for I in $BINDIFS
    do
        if [ -d /sys/class/net/$I ]; then
            brctl addif $BRNAME $I
            ifconfig $I 0.0.0.0 up
        fi
    done
}

bridge_stop() {
    echo "bridge_stop..."
    ifconfig $BRNAME down
    brctl delbr $BRNAME
}

# add bridge iface before bring up network
if [ ! -x /usr/sbin/brctl ]; then
    exit 0
fi

if [ $IFACE != $BRNAME ]; then
    exit 0
fi

case $MODE in
    start)
        bridge_start
        ;;
    stop)
        bridge_stop
        ;;
    *)
        exit 0
    ;;
esac
```

这里，我把bridge脚本放到了/usr/lib/scripts目录里，这样方便管理，那么这个脚本该怎么用呢？

```
# ls -la /etc/network/if-pre-up.d/bridge
lrwxrwxrwx 1 root root 23 Feb 12 13:30 /etc/network/if-pre-up.d/bridge -> /usr/lib/
scripts/bridge
# ls -la /etc/network/if-post-down.d/bridge
```

```
lrwxrwxrwx 1 root root      23 Feb 12 13:31 /etc/network/if-post-down.d/bridge -> /usr/lib/scripts/bridge
```

看，我们做了个符号连接，他就是用到这里！

## 警告

以上脚本运行在buildroot系统中，而不是主机系统里，亲，别放错地方！

## 配置interfaces

仅仅创建了bridge脚本还不够，我们还需要修改/etc/network/interfaces文件来触发他执行：

```
# cat /etc/network/interfaces
```

```
# Configure Loopback
auto lo
iface lo inet loopback

auto br0
iface br0 inet static
address 192.168.1.1
netmask 255.255.255.0
broadcast 192.168.1.255
```

这里桥接我们设置成静态的啦，因为对于lan侧的PC，我们的路由器必须提供一个固定的地址来进行访问才行。

## 配置dnsmasq

桥接接口总算是设置好了，可是，客户如何从路由器lan侧获取IP呢？显然，还是需要“自己动手，丰衣足食”了，那么就该dnsmasq登场了。

```
# cat /etc/dnsmasq.conf

conf-file=/etc/dnsmasq.conf
dhcp-authoritative
domain-needed
localise-queries
read-ethers
bogus-priv
expand-hosts
domain=lan
server=/lan/
dhcp-leasefile=/tmp/dhcp.leases
resolv-file=/tmp/resolv.conf.auto
stop-dns-rebind
rebind-localhost-ok
```

```

dhcp-range=lan,192.168.1.100,192.168.1.249,255.255.255.0,12h
no-dhcp-interface=eth1 eth2

address=/buildroot.lan/192.168.1.1
ptr-record=1.1.168.192.in-addr.arpa,buildroot.lan

```

哎，虽然有了配置文件，但是dnsmasq自己也没有长腿，我们还要给他添加一个自动运行的脚本：

```

# cat /etc/init.d/S50dnsmasq

#!/bin/sh
#
# Start the dnsmasq....
#

case "$1" in
start)
    echo "Starting dnsmasq..."
    /usr/sbin/dnsmasq -C /etc/dnsmasq.conf
    ;;
stop)
    echo -n "Stopping dnsmasq..."
    PID=`cat /var/run/dnsmasq.pid`
    kill -2 $PID
    rm /var/run/dnsmasq.pid
    ;;
restart|reload)
    "$0" stop
    "$0" start
    ;;
*)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
esac

exit $?

```

最后要强调的是，一定要给他加上可执行权限哦！

```
# chmod +x /etc/init.d/S50dnsmasq
```

## 功能验证

好了，一个崭新的“动物人”终于诞生了，接下来就该是“是骡子是马，牵出去溜溜了！”显然，单是一个“老章”恐怕看出来效果了，那么就让openwrt友情出演！

```
liunx@ubuntu:~/Work/OpenWrt/work/x86$ cat client.sh
```

```
#!/bin/bash -
set -o nounset                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel images/openwrt-x86-generic-vmlinuz \
    -hda images/openwrt-x86-generic-rootfs-ext4.img \
    -append "root=/dev/sda console=ttyS0" \
    -enable-kvm \
    -nographic \
    -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:02 \
    -net vde,vlan=0,sock=/tmp/lan1ctl
```

重点就在这里，通过“老章”的“线”连接上去！接下来./client.sh然后从“老章”获取IP看看，不过在这之前，我们还需要对openwrt做个小配置：

```
root@OpenWrt:/# cat /etc/config/network
# Copyright (C) 2006 OpenWrt.org
```

```
config interface loopback
    option ifname  lo
    option proto   static
    option ipaddr  127.0.0.1
    option netmask 255.0.0.0
```

```
config interface lan
    option ifname  eth1
    option type    bridge
    option proto   static
    option ipaddr  192.168.1.1
    option netmask 255.255.255.0
```

```
root@OpenWrt:/# /etc/init.d/network restart
```

我们只是把eth0从桥接设备中解放出来，然后重启网络配置。

```
root@OpenWrt:/# udhcpc -i eth0
udhcpc (v1.19.4) started
[ 35.887280] 8021q: adding VLAN 0 to HW filter on device eth0
Sending discover...
[ 37.891026] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Sending discover...
Sending select for 192.168.1.166...
Lease of 192.168.1.166 obtained, lease time 43200
udhcpc: ifconfig eth0 192.168.1.166 netmask 255.255.255.0 broadcast 192.168.1.255
udhcpc: setting default routers: 192.168.1.1
```

哈，成功了！

---

## 第 5 章 防火墙篇

防火墙的基本作用从他的名字就可以看出来，“墙”的作用不言而喻，大家都明白，就是防止别人进入，比如说世界八大奇迹之一的长城，就是一堵大墙，当然了，这是一个有形的“墙”，而我们讨论的是则是无形的“墙”，其实找个更直白的例子就是天朝的“大防火墙”，如果你想XXX的话，你就感受到他的存在了，剩下的你懂得。

好了，言归正传，防火墙的作用简单的概括就是数据过滤，他不仅过滤进入的数据，同时也过滤出去的数据。其实在网络世界中，事情更加复杂，防火墙还需要对数据进行转发等工作，这里我们一一讲述。

### 准备工作

有道是“磨刀不误砍材工”，在开始搭建防火墙之前，我们需要做一些准备工作。防火墙的搭建需要两个部分进行配置：

1.NetFilter

2.iptables

简单的描述一下二者的关系：NetFilter是用来执行防火墙规则的，iptables则是用来定义防火墙规则的。首先make menuconfig, 进入 Package Selection for the target → Networking applications 勾选“iptables”，保存退出。然后make linux-menuconfig:

```
[*] Networking support --->
  Networking options --->
    [*] Network packet filtering framework (Netfilter) --->
      [*] Advanced netfilter configuration
      Core Netfilter Configuration --->
        [*] Netfilter connection tracking support
        [*] "NFLOG" target support
        [*] "conntrack" connection tracking match support
        [*] "state" match support
      IP: Netfilter Configuration --->
        [*] IPv4 connection tracking support (required for NAT)
        [*] proc/sysctl compatibility with old connection tracking
        [*] IP tables support (required for filtering/masq/NAT)
        [*] Packet filtering
        [*] REJECT target support
        [*] Full NAT
        [*] MASQUERADE target support
        [*] NETMAP target support
        [*] REDIRECT target support
        [*] Basic SNMP-ALG support
        [*] Packet mangling
```

按照以上配置选择，最后保存退出并make。

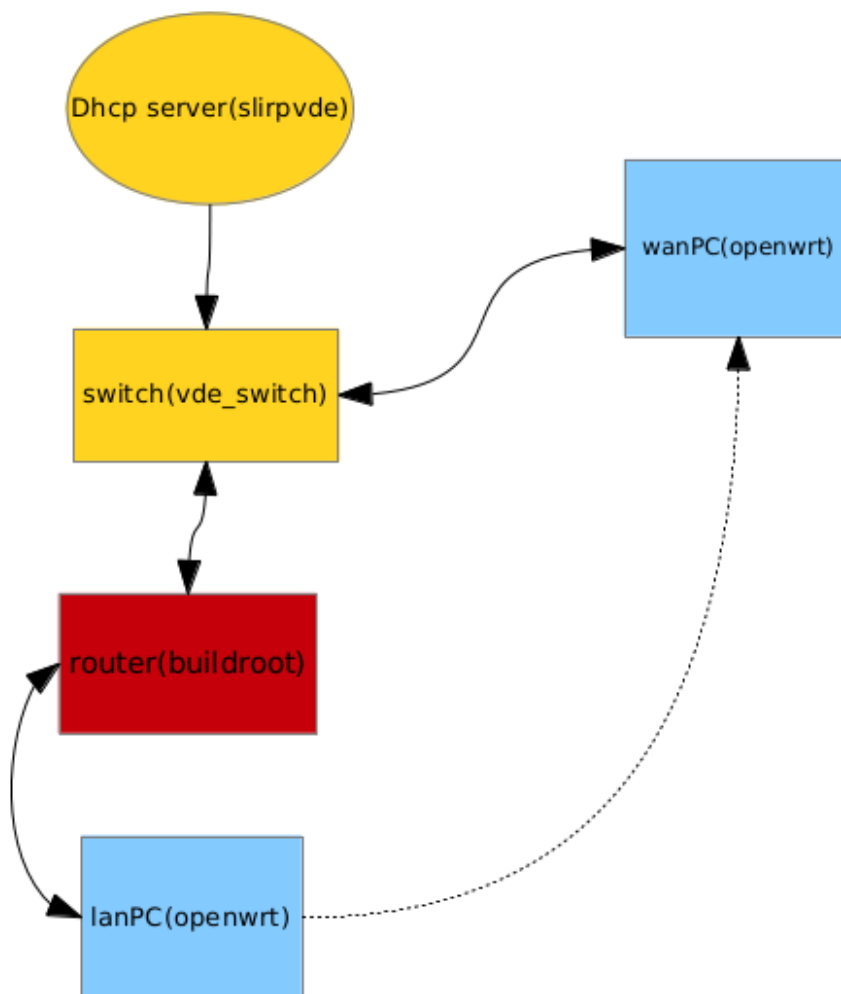
### 基本防火墙

好了，准备工作完成了，下面开始搭建防火墙了！



## 目标描述

下面请看我们的设计图：



黄色的部分是通过vde\_switch和slirpvde模拟的交换机网络，这里slirpvde起到分配IP地址的作用。红色部分就是我们的男主角“老章”了，蓝色部分就是由openwrt友情出演的wan侧主机和lan侧主机。

好，所有演员就位了，该是介绍剧情的时候了，我们的剧情很简单，那就是

1. 实现对lan/wan侧PC的访问的控制
2. 实现lan/wan侧PC的互访

## 创建交换机网络

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ cat custom/switch.sh
#!/bin/bash -
set -o nounset                                # Treat unset variables as an error
```

```
echo "begin switch..."
vde_switch --daemon --sock /tmp/switch --mgmt /tmp/switch.mgmt
slirpvde -d -s /tmp/switch -dhcp
echo "OK"
```

这个脚本很简单，我们就不做介绍了，唯一需要强调的是“--mgmt /tmp/switch.mgmt”，这个选项很重要哦，这里我们暂且留个悬念吧。

## 创建wan侧PC

```
liunx@ubuntu:~/Work/OpenWrt/work/x86$ cat wanPC.sh
#!/bin/bash -
set -o nounset                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel images/openwrt-x86-generic-vmlinuz \
    -hda images/openwrt-x86-generic-rootfs-ext4.img \
    -append "root=/dev/sda console=ttyS0" \
    -enable-kvm \
    -daemonize \
    -vnc :53 \
    -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:41 \
    -net vde,sock=/tmp/switch
```

好，运行这个脚本，让我们看看我们的模拟交换机网络是否工作正常：

```
liunx@ubuntu:~/Work/OpenWrt/work/x86$ ./wanPC.sh
```

### qemu vnc 端口号

关于qemuvnc端口号的设置，端口号 = 5900 + 定制的数字，例如“-vnc :53”，那么我们访问的vnc端口号就是5953。

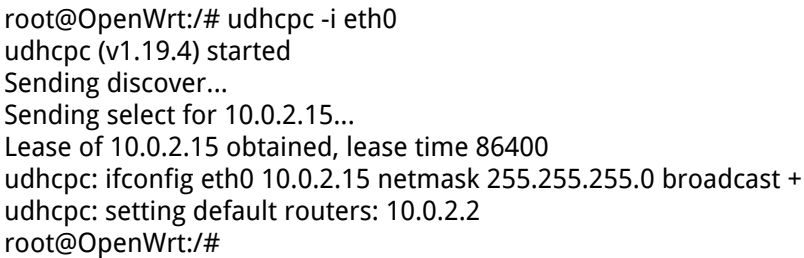
哎呀，啥输出都没有，但其实我们的qemu已经在后台悄悄地运行了！ 这里我们打开vnc来看看究竟。

```
liunx@ubuntu:~/Work/DocBook/Work/code-farmer-note$ vncviewer localhost:5953
```

```
VNC Viewer Free Edition 4.1.1 for X - built Feb 5 2012 20:01:21
Copyright (C) 2002-2005 RealVNC Ltd.
See http://www.realvnc.com for information on VNC.
```

```
Fri Feb 15 13:22:58 2013
CConn:    connected to host localhost port 5953
CConnection: Server supports RFB protocol version 3.8
CConnection: Using RFB protocol version 3.8
```

```
Fri Feb 15 13:23:12 2013
CConn:  Throughput 6666 kbit/s - changing to hextile encoding
CConn:  Throughput 6666 kbit/s - changing to full colour
CConn:  Using pixel format depth 24 (32bpp) little-endian rgb888
CConn:  Using hextile encoding
```



## 创建路由器

System configuration --->  
(tty1) Port to run a getty (login prompt) on

这样，我们就可以正常登录了。然后安装xvnc4viewer：

```
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ sudo apt-get install xvnc4viewer
liunx@ubuntu:~/Work/Buildroot/work/buildroot-2012.11/output/images$ cat router.sh
#!/bin/bash -
set -o nounset                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel images/bzImage \
    -hda images/rootfs.ext2 \
    -append "root=/dev/sda console=ttyS0" \
    -enable-kvm \
    -daemonize \
    -vnc :51 \
    -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:01 \
    -net vde,vlan=0,sock=/tmp/switch \
    -net nic,vlan=1,model=e1000,macaddr=52:54:00:12:34:11 \
    -net vde,vlan=1,sock=/tmp/lan1.ctl[] \
    -net nic,vlan=2,model=e1000,macaddr=52:54:00:12:34:12 \
    -net vde,vlan=2,sock=/tmp/lan2.ctl[] \
    -net nic,vlan=3,model=e1000,macaddr=52:54:00:12:34:13 \
    -net vde,vlan=3,sock=/tmp/lan3.ctl[] \
    -net nic,vlan=4,model=e1000,macaddr=52:54:00:12:34:14 \
    -net vde,vlan=4,sock=/tmp/lan4.ctl[]
```

这里我们定义了eth0为外网接口，验证方法同上。

## 创建lan侧PC

好了，路由器有了，我们接下来再添加一个PC连接到路由器的lan侧。

```
liunx@ubuntu:~/Work/OpenWrt/work/x86$ cat lanPC.sh
#!/bin/bash -
set -o nounset                # Treat unset variables as an error

/usr/local/bin/qemu-system-i386 \
    -m 64M \
    -kernel images/openwrt-x86-generic-vmlinuz \
    -hda images/openwrt-x86-generic-rootfs-ext4.img \
    -append "root=/dev/sda console=ttyS0" \
    -enable-kvm \
    -daemonize \
    -vnc :52 \
    -net nic,vlan=0,model=e1000,macaddr=52:54:00:12:34:31 \
    -net vde,vlan=0,sock=/tmp/lan1.ctl
```

这里，我们选择路由器中任意一个lanx.ctl接口就可以了。

接下来，我们验证一下是否能从路由器获取IP地址：

```
root@OpenWrt:/# udhcpc -i eth0
udhcpc (v1.19.4) started
Sending discover...
Sending select for 192.168.1.213...
Lease of 192.168.1.213 obtained, lease time 43200
udhcpc: ifconfig eth0 192.168.1.213 netmask 255.255.255.0 broadcast 192.168.1.255
udhcpc: setting default routers: 192.168.1.1
```

OK，没问题！

## 创建防火墙规则

下面到了关键的环节，我们要创建防火墙规则了。

### 接口

```
# Private interface
IF_PRV=br0
IP_PRV=192.168.1.1
NET_PRV=192.168.1.0/24

# Public interface
IF_PUB=eth0
IP_PUB=10.0.0.1
NET_PUB=10.0.0.0/24

# Others
ANYWHERE=0.0.0.0/0
```

这里我们需要区分的是公共接口和私有接口，公共接口就是eth0，私有接口就是我们的br0了。公共接口和外部交换机相连，而私有接口则和lan侧的主机相连。

### 设置默认表策略

```
# iptables -nvL
Chain INPUT (policy ACCEPT 388 packets, 27432 bytes)
  pkts bytes target    prot opt in   out   source          destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target    prot opt in   out   source          destination

Chain OUTPUT (policy ACCEPT 386 packets, 26280 bytes)
  pkts bytes target    prot opt in   out   source          destination
```

路由表默认的策略是ACCEPT，即网络包没有匹配任何规则时，默认接受所有的网络包，

```

root@OpenWrt:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:01
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:144 errors:0 dropped:0 overruns:0 frame:0
          TX packets:867 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:12403 (12.1 KiB)  TX bytes:71824 (70.1 KiB)

root@OpenWrt:~# ping -c 3 10.0.2.15
PING 10.0.2.15 (10.0.2.15): 56 data bytes
64 bytes from 10.0.2.15: seq=0 ttl=64 time=1.316 ms
64 bytes from 10.0.2.15: seq=1 ttl=64 time=1.219 ms
64 bytes from 10.0.2.15: seq=2 ttl=64 time=1.303 ms

--- 10.0.2.15 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.219/1.306/1.303 ms
root@OpenWrt:~#

root@OpenWrt:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:01
          inet addr:192.168.1.219  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15473 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15473 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1265800 (1.2 MiB)  TX bytes:1262604 (1.2 MiB)

root@OpenWrt:~# ping -c 3 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=0.977 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=1.205 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=1.114 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.977/1.090/1.205 ms
root@OpenWrt:~#
  
```

这样的路由器简直就成了筛子了，显然不是我们想要的结果，这里我们设定一个严格而保守的策略，默认不接受任何包。

```

# iptables -P INPUT DROP
# iptables -P OUTPUT DROP
# iptables -P FORWARD DROP
# iptables -nvL
Chain INPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
  
```

这时我们再次进行PING测试：

```

# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15): 56 data bytes
ping: sendto: Operation not permitted
  
```

```

root@OpenWrt:~# ping 10.0.2.16
PING 10.0.2.16 (10.0.2.16): 56 data bytes
^C
--- 10.0.2.16 ping statistics ---
  
```

103 packets transmitted, 0 packets received, 100% packet loss

```
root@OpenWrt:/# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

不行了哟！

在开始添加规则前，我们也要做一个清理工作，清除以前的规则：

```
# iptables -F -t filter
# iptables -F -t nat
# iptables -F -t mangle
# iptables -X
```

## 开启路由功能

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

首先，让我们打开内核的数据转发功能。接下来设置转发规则：

```
# iptables -A FORWARD -i $IF_PUB -o $IF_PRV -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -i $IF_PRV -o $IF_PUB -j ACCEPT
```

从转发规则可以看出，我们限制外部的数据转发，无条件的接受内部网络对外部的数据转发。

## 信任内部网络

由于我们严格的DROP策略，默认情况下，我们连内部回路都不能用了：

```
# ifconfig lo
lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  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:0
      RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

# ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
ping: sendto: Operation not permitted
```

我们不信任别人，至少要信任自己吧，是不是？

```
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A OUTPUT -o lo -j ACCEPT
# ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.105 ms
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.160 ms
64 bytes from 127.0.0.1: seq=2 ttl=64 time=0.164 ms
c
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.105/0.143/0.164 ms
```

除此以外，我们还要信任自己的私有网络：

```
# iptables -A INPUT -i $IF_PRV -s $NET_PRV -j ACCEPT
# iptables -A OUTPUT -o $IF_PRV -d $NET_PRV -j ACCEPT
```

接下来，我们要为私有网络添加路由规则，已完成我们的第二个目标，即私有网络能够访问外部网络。

```
# iptables -t nat -A POSTROUTING -s $NET_PRV -o $IF_PUB -j SNAT --to $IP_PUB
```

这里，我们需要网络地址NAT，并且我们的私有地址是无法被外部网络认证的，因此我们需要把私有地址转换成公网地址，这样才能与外部网络传递数据，所以这种对源地址做转换的模式叫源地址转换SNAT。

## 开启服务

我们的路由器还需要提供一些基本的服务，比如DHCP服务，同时为了方便调试，我们还要提供ICMP服务，所以我们还需要添加规则：

```
# iptables -A INPUT -p icmp -j ACCEPT
# iptables -A INPUT -p udp --dport 67:68 --sport 67:68 -j ACCEPT
```

## 总结

好了，我们的简单防火墙已出具成效了，下面我们要把他们写成自动运行脚本，以方便使用。

```
# cat /etc/init.d/S60firewall
#!/bin/sh
# Copyright (c) 2005
#
# Author: David Mair
#
```



```
# /etc/init.d/firewall
#
# Moditfied by Liunx

# DEFAULT POLICY
SetDefaultPolicy() {
    # Drop everything
    iptables -P INPUT DROP
    iptables -P OUTPUT DROP
    iptables -P FORWARD DROP
}

# FLUSH TABLES
FlushTables() {
    iptables -F -t nat
    iptables -F -t mangle
    iptables -F -t filter
    iptables -X
}

# ROUTING
EnableRouting() {
    echo 1 > /proc/sys/net/ipv4/ip_forward
}

DisableRouting() {
    echo 0 > /proc/sys/net/ipv4/ip_forward
}

# FORWARDING
SetForwardingRules() {
    iptables -A FORWARD -i $IF_PUB -o $IF_PRV -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
    iptables -A FORWARD -i $IF_PRV -o $IF_PUB -j ACCEPT
}

# LOOPBACK
SetLoopbackRules() {
    # Allow everything
    iptables -A INPUT -i lo -j ACCEPT
    iptables -A OUTPUT -o lo -j ACCEPT
}

# PRIVATE INTERFACES
SetPrivateInterfaceRules() {
    # Allow everything
    iptables -A INPUT -i $IF_PRV -s $NET_PRV -j ACCEPT
    iptables -A OUTPUT -o $IF_PRV -d $NET_PRV -j ACCEPT
}
```

```
# PUBLIC INTERFACES
SetPublicInterfaceRules() {
  iptables -A INPUT -i $IF_PUB -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
  iptables -A OUTPUT -o $IF_PUB -j ACCEPT
}

# SOURCE NAT
EnableSourceNAT() {
  # Then source NAT everything else
  iptables -t nat -A POSTROUTING -s $NET_PRV -o $IF_PUB -j SNAT --to $IP_PUB
}

# Various ICMP
SetICMP_Open() {
  iptables -A INPUT -p icmp -j ACCEPT
}

# enable dhcp for private network
SetDHCP_Open() {
  iptables -A INPUT -p udp --dport 67:68 --sport 67:68 -j ACCEPT
}

# Drop them all
SetDropRules() {
  # Reset tcp connection attempts on all other ports
  # This is the standard TCP behaviour for a closed port. Reading
  # suggests there is no value in stealthing ports and since some are
  # open on this host it doesn't seem to matter. Therefore, let's be a
  # good TCP citizen
  iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset
}

# SCRIPT ENTRY POINT

echo -n "Firewall configuration..."
echo $1

# ENVIRONMENT

# Private interface
IF_PRV=br0
IP_PRV=192.168.1.1
NET_PRV=192.168.1.0/24

# Public interface
IF_PUB=eth0
IP_PUB=10.0.2.16
NET_PUB=10.0.2.0/24
```

```
# Others
ANYWHERE=0.0.0.0/0

# COMMAND LINE

case "$1" in
start)
    SetDefaultPolicy
    FlushTables
    EnableRouting
    SetForwardingRules
    SetLoopbackRules
    SetPrivateInterfaceRules
    SetPublicInterfaceRules
    EnableSourceNAT

    SetDHCP_Open
    SetICMP_Open
    SetDropRules
    ;;

stop)
    SetDefaultPolicy
    FlushTables

    SetPrivateInterfaceRules
    SetPublicInterfaceRules
    ;;

restart)
    $0 stop
    $0 start
    ;;

*)
    ;;
esac
```

---

## 第 6 章 防火墙高级篇

---

## 第 7 章 虚拟私有网络

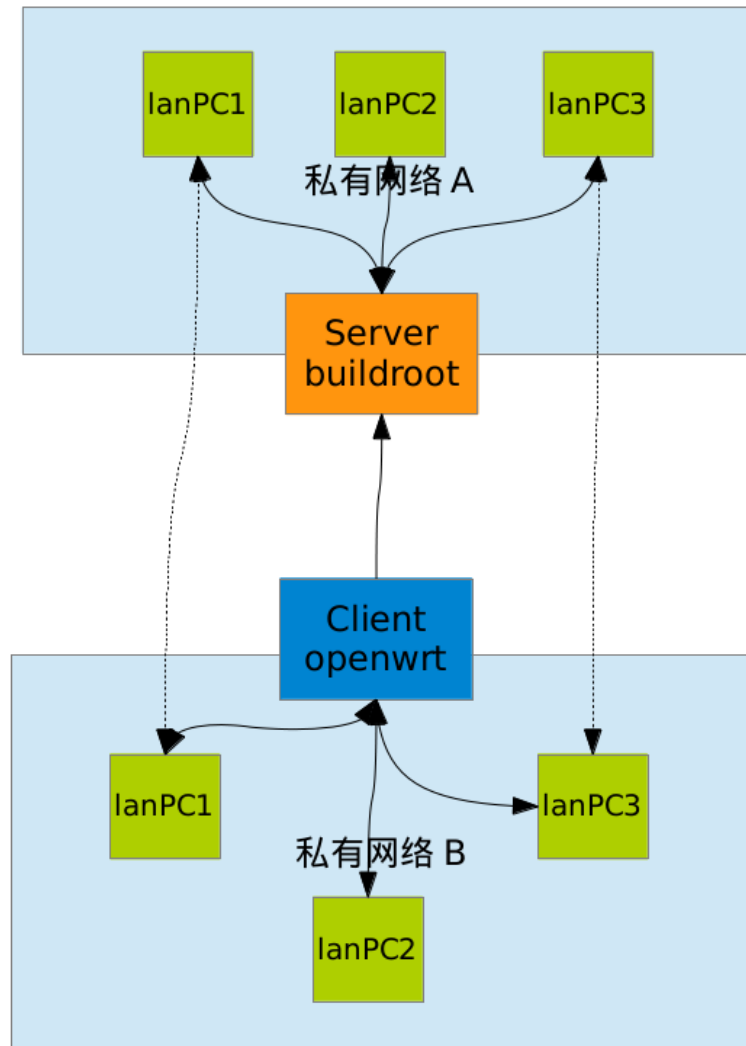
虚拟私有网络就是我们常说的VPN，如果你对VPN一无所知的话，那么我们打一个简单的比方就是--“深入敌人内部”。显然，在“敌人内部”，我们就需要特殊的渠道进行情报交流了，这个渠道就是“IPSec”和“l2tp”了，并且两者需要配置使用才能发挥最大的功效。“IPSec”用来对数据进行加密，“l2tp”则用来建立通道。

### L2TP连接

其实，通过l2tp服务，我们就可以建立VPN通道了，但是由于l2tp没有加密，所以我们的通信是暴露的，很容易被窃取的，所以有安全风险的，但是实验中我们可以通过l2tp来了解VPN究竟是如何工作的。

### 准备工作

搭建基于l2tp的VPN连接，我们需要至少一个服务端 -- 拥有公共的地址，和一个客户端来创建连接。网络拓扑图如下：



从图中可以看出，私有网络A和B是两个独立的内部网络，无法互访，这时，我们需要通过VPN来让他们实现互访，这样，无论两个网络在物理上相隔万水千山，但在实际通信中确实近在咫尺，这也就是VPN的作用之所在了。

好，接下来，让我们看看如何把我们的buildroot的系统，改造成l2tp服务器吧

## 配置内核

首先，让我们配置内核，linux支持l2tp协议的，但是这里我们使用的是xl2tpd，他没有使用linux的内核模块，而是把协议处理做到了用户空间，然而我们最终需要ppp完成拨号连接，而pppd需要内核模块的支持，所以“Let's do it!”

```
Device Drivers --->
[*] Network device support --->
    [*] PPP (point-to-point protocol) support
    [*] PPP support for async serial ports
```

我们至少打开这两个选项就可以了，其他的选项以后也许用得着。

## 配置l2tp服务端

首先选取xl2tpd软件包和ppp包：

```
Package Selection for the target --->
Networking applications --->
    [*] pppd
    [*] xl2tp
```

服务端我们采用buildroot，配置文件如下：

```
# cat /etc/xl2tpd/xl2tpd.conf
[global]
port = 1701
auth file = /etc/xl2tpd/xl2tp-secrets
access control = no

[lns default]
exclusive = yes
ip range = 112.168.254.202-112.168.254.210
hidden bit = no
local ip = 112.168.254.200
length bit = yes
;require authentication = yes
name = VersaLink
;ppp debug = yes
pppoptfile = /etc/ppp/options.xl2tpd

# cat /etc/ppp/options.xl2tpd
#
```

```
lock
noauth
debug
#dump
logfd 2
logfile /var/log/xl2tpd.log
ncccp
novj
novjccomp
nopcomp
noaccomp

# cat /etc/ppp/chap-secrets
#USERNAME PROVIDER PASSWORD IPADDRESS
admin      *      admin      *
```

## 配置l2tp客户端

客户端由openwrt友情出演，上配置文件：

```
root@OpenWrt:/# cat /etc/xl2tpd/xl2tpd.conf
[global]
port = 1701
auth file = /etc/xl2tpd/xl2tp-secrets
access control = no

;[lns default]
;exclusive = yes
;ip range = 192.168.254.202-192.168.254.210
;lac = 10.0.1.2
;hidden bit = no
;local ip = 192.168.254.200
;length bit = yes
;refuse authentication = yes
;name = VersaLink
;ppp debug = yes
;pppoptfile = /etc/ppp/options.xl2tpd

[lac left]
lns = 10.0.2.16
require authentication = yes
name = VersaLink
;ppp debug = yes
pppoptfile = /etc/ppp/options.xl2tpd

root@OpenWrt:/# cat /etc/ppp/options.xl2tpd
#

name admin
password admin
lock
noauth
```



```
debug
#dump
logfd 2
logfile /var/log/xl2tpd.log
noccp
novj
novjccomp
nopcomp
noaccomp
```

## 警告

注意，lns就是客户端将要连接的服务端地址，这里根据实际情况填写，请勿“生搬硬套”，而是“照猫画虎”，变通一下才对！

## 开始实验！

好了，准备工作完成了，下面开始实验了，你准备好了吗？

首先让我们打开服务端：

```
# mkdir /var/run/xl2tpd
# mknod /dev/ppp c 108 0
# xl2tpd -D &
# xl2tpd[113]: setsockopt recvref[30]: Protocol not available
xl2tpd[113]: This binary does not support kernel L2TP.
xl2tpd[113]: xl2tpd version xl2tpd-1.3.1 started on buildroot PID:113
xl2tpd[113]: Written by Mark Spencer, Copyright (C) 1998, Adtran, Inc.
xl2tpd[113]: Forked by Scott Balmos and David Stipp, (C) 2001
xl2tpd[113]: Inherited by Jeff McAdams, (C) 2002
xl2tpd[113]: Forked again by Xelerance (www.xelerance.com) (C) 2006
xl2tpd[113]: Listening on IP address 0.0.0.0, port 1701
```

这里，我们首先创建/var/run/xl2tpd目录，然后创建/dev/ppp设备节点，最后以调试模式运行。

接下来创建连接：

```
root@OpenWrt:/# /etc/init.d/xl2tpd restart
root@OpenWrt:/# echo "c left" > /var/run/xl2tpd/l2tp-control
root@OpenWrt:/# ifconfig
eth0  Link encap:Ethernet HWaddr 12:34:56:78:1A:00
      inet addr:10.0.2.16 Bcast:10.0.2.255 Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:4261 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4304 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:267068 (260.8 KiB) TX bytes:261204 (255.0 KiB)

lo    Link encap:Local Loopback
```

```
inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:20 errors:0 dropped:0 overruns:0 frame:0
TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:3160 (3.0 KiB) TX bytes:3160 (3.0 KiB)
```

```
ppp0 Link encap:Point-to-Point Protocol
inet addr:112.168.254.202 P-t-P:112.168.254.200 Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:3 errors:0 dropped:0 overruns:0 frame:0
TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:30 (30.0 B) TX bytes:30 (30.0 B)
```

openwrt有自启动脚本，所以用起来比较方便一点，如果看到ppp0了，那么我们的连接就创建成功了！

接下来让我们看看服务端有哪些反映吧：

```
# xl2tpd[113]: control_finish: Peer requested tunnel 12405 twice, ignoring second one.
xl2tpd[113]: Connection established to 10.0.2.16, 1701. Local: 43197, Remote: 12405 (ref=0/0).
LNS session is 'default'
xl2tpd[113]: control_finish: Warning: Peer did not specify transmit speed
xl2tpd[113]: start_pppd: I'm running:
xl2tpd[113]: "/usr/sbin/pppd"
xl2tpd[113]: "passive"
xl2tpd[113]: "nodetach"
xl2tpd[113]: "112.168.254.200:112.168.254.202"
xl2tpd[113]: "name"
xl2tpd[113]: "VersaLink"
xl2tpd[113]: "file"
xl2tpd[113]: "/etc/ppp/options.xl2tpd"
xl2tpd[113]: "ipparam"
xl2tpd[113]: "10.0.2.16"
xl2tpd[113]: "/dev/pts/0"
xl2tpd[113]: Call established with 10.0.2.16, Local: 14252, Remote: 62314, Serial: 1

# ifconfig ppp0
ppp0 Link encap:Point-to-Point Protocol
inet addr:112.168.254.200 P-t-P:112.168.254.202 Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:3 errors:0 dropped:0 overruns:0 frame:0
TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:30 (30.0 B) TX bytes:30 (30.0 B)
```

## 创建IPSEC连接

ipsec常见的软件有ipsec-tools、openswan、strongswan、openvpn等，这里我们先来介绍一下如何使用ipsec-tools来搭建ipsec连接。

## 准备工作

其实一路走过来，大家也总结出一个规律--“先配置内核，再配置用户空间程序”，将协议做到内核中，可以提高效率，充分发挥linux内核作用，所以让我先看看如何启用内核中的ipsec功能吧。

```
[*] Networking support --->
    Networking options --->
        <*> PF_KEY sockets
        <M> IP: ESP transformation
        <M> IP: IPsec transport mode
        <M> IP: IPsec tunnel mode
    *- Cryptographic API --->
        {*} HMAC support
        *- CRC32c CRC algorithm
        {*} MD5 digest algorithm
        {*} SHA1 digest algorithm
        <*> Tiger digest algorithms
        *- AES cipher algorithms
            {M} AES cipher algorithms (i586)
            <M> AES cipher algorithms (AES-NI)
            <M> Blowfish cipher algorithm
            {M} DES and Triple DES EDE cipher algorithms
```

配置内核我们做了两件事，一是添加协议支持，二是添加加密算法。接下来让我们选取ipsec-tools吧。

```
Package Selection for the target --->
    Networking applications --->
        [*] ipsec-tools
        [*] Enable racoonctl(8)
        [*] Enable NAT-Traversal
        [*] Enable IKE fragmentation
        [*] Enable DPD (Dead Peer Detection)
        [*] Enable statistics logging function
        [*] Enable readline input support
```

## 配置阶段

OK，又到了配置阶段了，闲言少叙，上配置：

```
# cat /etc/racoon.conf

path pre_shared_key "/etc/racoon/psk.txt";

remote 10.0.2.16 {
    exchange_mode main,aggressive;
    proposal {
        encryption_algorithm 3des;
```

```

        hash_algorithm sha1;
        authentication_method pre_shared_key;
        dh_group 2;
    }
}

sainfo address 172.20.1.0/24 any address 192.168.22.0/24 any {
    pfs_group 2;
    lifetime time 1 hour ;
    encryption_algorithm 3des, blowfish 448, rijndael ;
    authentication_algorithm hmac_sha1, hmac_md5 ;
    compression_algorithm deflate ;
}
# cat /etc/racoon/psk.txt
10.0.2.16 123456
# cat /etc/ipsec-tools.conf
flush;
spdf flush;

spdadd 172.20.1.0/24 192.168.22.0/24 any -P out ipsec
        esp/tunnel/172.27.1.165-172.27.1.169/require;

spdadd 192.168.22.0/24 172.20.1.0/24 any -P in ipsec
        esp/tunnel/172.27.1.169-172.27.1.165/require;
#
# cat /etc/init.d/S50racoon
#!/bin/sh
#
# Start the racoon based ipsec....
#

start() {
    mkdir -m 0700 -p /var/racoon
    [ -f /etc/ipsec-tools.conf ] && /usr/sbin/setkey -f /etc/ipsec-tools.conf
    start-stop-daemon -x /usr/sbin/racoon -b -m -p /var/run/racoon.pid -S -- -f /etc/racoon.conf
}

stop() {
    start-stop-daemon -x /usr/sbin/racoon -s 2 -p /var/run/racoon.pid -K
}

case "$1" in
    start)
        echo "Starting racoon..."
        start
        ;;
    stop)
        echo "Stopping racoon..."
        stop
        ;;
    restart|reload)
        "$0" stop
        "$0" start
        ;;

```

```
*)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
esac

exit $?
```

---

## 第 8 章 Quality of Service (QoS)

QoS翻译过来就叫做服务质量，显然，在资源一定的情况下，如何提高服务质量，就是我们本章需要解决的问题。不知大家是否有在宿舍共享上网的痛苦经历，如果有些“不法分子”开启迅雷等下载工具的时候，是否感觉网络异常的慢呢？呵呵，想来确实可气，要是能够对某些不法分子进行惩戒就好了，那么QoS就派上用场了！

### 准备工作

QoS工作的核心还是linux内核，也就是说网络资源分配的执行终端就是内核，而我们只是通过用户空间程序来给内核制定工作内容和创建规则。

首先让我们打来自内核的QoS功能。

```
[*] Networking support --->
Networking options --->
[*] QoS and/or fair queueing --->
    <M> Token Bucket Filter (TBF)
    <M> Class Based Queueing (CBQ)
    <M> Hierarchical Token Bucket (HTB)
```

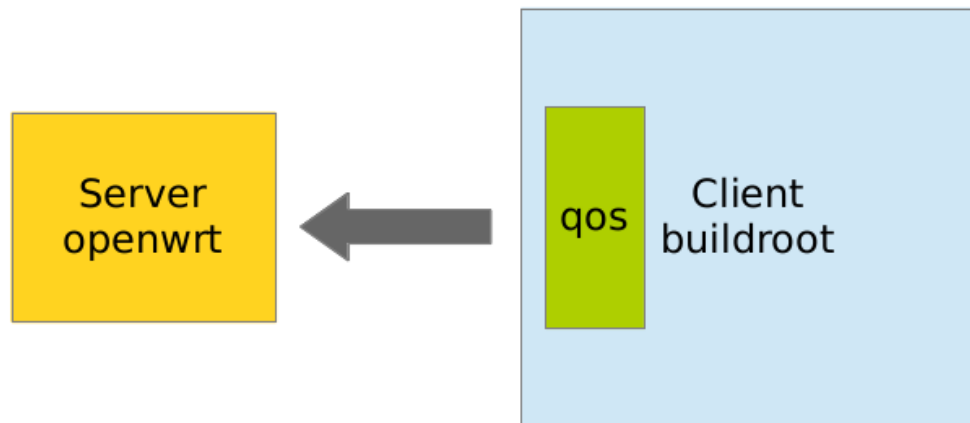
接下来让我们选择用户空间程序：

```
Package Selection for the target --->
Networking applications --->
[*] iperf
[*] iproute2
```

tc用来操作内核QoS模块的，而iperf则是用来测试网络性能的工具。

### 实验阶段

好了，现在进入实验阶段，下面看看网络拓扑图：



嗯？这次openwrt倒是“反客为主”，这是为什么呢？因为linux的QOS只能限制下行速度，也就是说只能限制发出的速度，而不能限制输入的速度，简单的比方就是“病从口入，祸从口出”！

## TBF简单限速

如果我们只是简单得限制上行速度，那么TBF-(Token Bucket Filter)就是很好的选择，因为他使用简单，并且很精确。那么接下来，就让我们看看TBF是如何产生效果的吧。

首先在openwrt服务端开启iperf服务：

```
# iperf -s
```

```
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
```

接下来，我们下不急于设置QOS规则，我们先“裸跑”一下，看看没有限速的情况下，我们的网络情况如何？

```
# iperf -c 10.0.2.16
```

```
-----
Client connecting to 10.0.2.16, TCP port 5001
TCP window size: 19.6 KByte (default)
-----
[ 3] local 10.0.2.15 port 40529 connected with 10.0.2.16 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.2 sec  3.88 MBytes  3.19 Mbits/sec
```

```
+++++
```

```
root@OpenWrt:/# iperf -s
```

```
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 10.0.2.16 port 5001 connected with 10.0.2.15 port 40529
[ ID] Interval   Transfer   Bandwidth
[ 4]  0.0-10.8 sec 3.88 MBytes 3.02 Mbits/sec
```

哈，vde网络的速度还行，当然和物理网络相比还是有一定差距的，好，接下来让我们看看QoS后，是个什么情况吧！

```
# tc qdisc add dev eth0 root tbf rate 60kbit latency 50ms burst 1540
# iperf -c 10.0.2.16
```

```
-----
Client connecting to 10.0.2.16, TCP port 5001
TCP window size: 19.6 KByte (default)
-----
[ 3] local 10.0.2.15 port 40530 connected with 10.0.2.16 port 5001
[ ID] Interval   Transfer   Bandwidth
[ 3]  0.0-20.1 sec 128 KBytes 52.1 Kbits/sec
```

```
+++++
```

```
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 10.0.2.16 port 5001 connected with 10.0.2.15 port 40529
[ ID] Interval   Transfer   Bandwidth
[ 4]  0.0-10.8 sec 3.88 MBytes 3.02 Mbits/sec
[ 4] local 10.0.2.16 port 5001 connected with 10.0.2.15 port 40530
[ 4]  0.0-22.6 sec 128 KBytes 46.3 Kbits/sec
```

嗯，效果很明显，很不错！



---

## 附录 A. 常见问题FAQ

这里，我们列出了常见问题和解决办法。减轻大家重复的劳动。

### openwrt pppd 段错误

如果出现“root@OpenWrt:/# [ 15.761824] pppd[1456]: segfault at 0 ip 08065f90 sp bf8dc9f0 error 4 in pppd[8048000+3b000]”的提示，那么我们需要修改/etc/ppp/options.xl2tpd文件。

```
root@OpenWrt:/etc/ppp# cat /etc/ppp/options.xl2tpd
#

lock
auth

debug
# We MUST cancel dump, or we'll get segfault

#dump

# CCP seems to confuse Android clients, better turn it off
noccp
novj
novjccomp
nopcomp
noaccomp
require-mschap
require-mschap-v2
ms-dns 172.17.2.1
lcp-echo-interval 120
lcp-echo-failure 10
idle 1800
connect-delay 5000
nodefaultroute
noipdefault

proxyarp
mtu 1400
mru 1400
```

这里我们注释掉“dump”选项即可。

### failed to initialize KVM: Device or resource busy No accelerator found!

如果你已经运行了virtualbox，那么你很可能会遇到这个问题，很不幸，Vbox有点“独”了，kvm被占用了，解决方法就是关掉Vbox或者取消“-enable-kvm”这一项吧。

---

# 参考文献

## 第一章

- <http://ubuntu.com.cn>Ubuntu官方网站，大家可以从这里找到系统镜像和安装方法。
- <http://xubuntu.org/>Xubuntu官方网站。
- <http://mirrors.163.com/> 网易提供的镜像站点，速度很快哦！
- <http://wiki.qemu.org/>qemu官方网站，大家这里可以下载代码。

## 第二章

- [http://wiki.virtualsquare.org/wiki/index.php/Main\\_Page](http://wiki.virtualsquare.org/wiki/index.php/Main_Page)ve官网，里面很多的tutorials值得参考。

`Iptables_for_newbies` [[http://en.gentoo-wiki.com/wiki/Iptables\\_for\\_newbies](http://en.gentoo-wiki.com/wiki/Iptables_for_newbies)] -- 来自 gentoo的wiki教新手配置防火墙，简单直白易懂，实战性很强。

`simple-firewall-configuration-using-netfilteriptables` [<http://www.novell.com/communities/node/3710/simple-firewall-configuration-using-netfilteriptables>] -- 来自Novell的技术文章，简单易懂，老少皆宜。

`Simple_stateful_firewall` [[https://wiki.archlinux.org/index.php/Simple\\_stateful\\_firewall](https://wiki.archlinux.org/index.php/Simple_stateful_firewall)] -- 来自archlinux的wiki。