下一代基站承载子系统移植记录

版本历史及修订说明

| 版本 | 日期 | 拟制人/修改人 | 版本修订说明 | 审核人 |
| --- | --- | --- | --- | --- |
| 0.1 | 2017.01.20 | Wuh | 初版 |  |
| 0.2 | 2017.12.14 | Wuh | 添加真实环境下基站和路由器的配置说明，见7.6、8.11 |  |
| 0.3 | 2018.01.11 | Wuh | 添加实际测试中出现的问题，见12 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[下一代基站承载子系统移植记录 1](#_Toc503787895)

[1 承载子系统主要模块简介 3](#_Toc503787896)

[1.1 TCP/IP协议栈支持 3](#_Toc503787897)

[1.2 时间管理功能块（ntp） 3](#_Toc503787898)

[1.3 语音承载功能块（VoiceBear） 3](#_Toc503787899)

[1.4 gre tunnel功能块（tunnel） 3](#_Toc503787900)

[1.5 vpn功能块（l2tp、ppp） 3](#_Toc503787901)

[1.6 动态路由功能块（ospf） 3](#_Toc503787902)

[1.7 与B1接口模块（B1AP） 3](#_Toc503787903)

[1.8 与PDC交互的模块（gre） 3](#_Toc503787904)

[1.8.1 基站侧实现如下 3](#_Toc503787905)

[1.8.2 PDC侧实现如下 4](#_Toc503787906)

[1.9 小结 4](#_Toc503787907)

[2 承载数据发送流程 5](#_Toc503787908)

[2.1 基站B1向交换发送数据的接口 5](#_Toc503787909)

[2.2 基站BSC向备板或BSR发送数据的接口 5](#_Toc503787910)

[2.3 BSC通用消息发送接口 5](#_Toc503787911)

[2.4 ICP模块报文监听模块(SocketListenThread) 6](#_Toc503787912)

[2.5 ICP本地消息客户端入口 icpFunc 6](#_Toc503787913)

[2.6 基站各类ICP通道监听的端口 8](#_Toc503787914)

[2.7 小结 8](#_Toc503787915)

[3 网络中一帧数据包格式 8](#_Toc503787916)

[4 数据结构 8](#_Toc503787917)

[4.1 IP头 8](#_Toc503787918)

[4.2 Gre头 9](#_Toc503787919)

[4.3 PPP帧 10](#_Toc503787920)

[4.4 L2tp头 10](#_Toc503787921)

[4.5 UDP头 11](#_Toc503787922)

[5 一个ip数据包如何在tcp/ip协议栈中传输 12](#_Toc503787923)

[6 UDP编程步骤 12](#_Toc503787924)

[7 gre组网 13](#_Toc503787925)

[7.1 Gre协议概要 13](#_Toc503787926)

[7.2 gre组网模型 14](#_Toc503787927)

[7.3 数据发送和接收流程 15](#_Toc503787928)

[7.4 如何配置隧道 16](#_Toc503787929)

[7.4.1 红帽子与路由器之间配置模型 17](#_Toc503787930)

[7.4.2 开发板与路由器之间配置模型 17](#_Toc503787931)

[7.4.3 两台主机之间通过2个路由器实现互联模型 18](#_Toc503787932)

[7.5 移植方法 19](#_Toc503787933)

[7.6 真实环境调试gre+ospf 19](#_Toc503787934)

[7.6.1 环境说明 19](#_Toc503787935)

[7.6.2 基站配置 20](#_Toc503787936)

[7.6.3 思科路由器配置 20](#_Toc503787937)

[7.6.4 调试说明 21](#_Toc503787938)

[8 L2TP组网 21](#_Toc503787939)

[8.1 概要 21](#_Toc503787940)

[8.2 两种典型的L2TP隧道模式 22](#_Toc503787941)

[8.3 Pc直接接入LNS（相当于基站接入LNS的模型） 22](#_Toc503787942)

[8.4 客户端和服务器隧道建立的过程 24](#_Toc503787943)

[8.4.1 控制连接的建立 25](#_Toc503787944)

[8.4.2 会话连接的建立 26](#_Toc503787945)

[8.4.3 保活 26](#_Toc503787946)

[8.4.4 会话和隧道的拆除 26](#_Toc503787947)

[8.5 隧道验证过程 27](#_Toc503787948)

[8.6 PPP协议栈 27](#_Toc503787949)

[8.7 L2tp组网模型 28](#_Toc503787950)

[8.8 数据发送和接收流程 29](#_Toc503787951)

[8.9 移植方法 31](#_Toc503787952)

[8.10 xl2tpd配置和运行 33](#_Toc503787953)

[8.10.1 Server配置 33](#_Toc503787954)

[8.10.2 Client配置 34](#_Toc503787955)

[8.10.3 运行xl2tpd 35](#_Toc503787956)

[8.10.4 配置路由 35](#_Toc503787957)

[8.10.5 crontab配置 35](#_Toc503787958)

[8.11 真实环境调试vpn+ospf 35](#_Toc503787959)

[8.11.1 环境说明 35](#_Toc503787960)

[8.11.2 基站配置 36](#_Toc503787961)

[8.11.3 思科路由器配置 37](#_Toc503787962)

[8.11.4 思科路由器上查看vpn调试日志 38](#_Toc503787963)

[8.11.5 思科路由器上查看ospf调试日志 38](#_Toc503787964)

[8.11.6 Wireshark抓包 39](#_Toc503787965)

[9 ospf模块 40](#_Toc503787966)

[9.1 代码、编译 40](#_Toc503787967)

[9.2 配置文件 41](#_Toc503787968)

[9.2.1 #zebra.conf 41](#_Toc503787969)

[9.2.2 #ospfd.conf 41](#_Toc503787970)

[10 最终下一代基站中运行的各模块代码路径和编译方法 41](#_Toc503787971)

[10.1 gre、ospf、vpn的初始化代码 41](#_Toc503787972)

[10.2 gre模块的内核模块代码 42](#_Toc503787973)

[10.3 Ospf模块的代码 42](#_Toc503787974)

[10.4 vpn模块ppp协议栈内核模块代码 42](#_Toc503787975)

[10.5 vpn模块pppd用户态代码 42](#_Toc503787976)

[10.6 vpn模块的l2tp协议栈代码 42](#_Toc503787977)

[10.7 打包bear\_install.bin 43](#_Toc503787978)

[11 打包说明 43](#_Toc503787979)

[12 实际调试中问题 43](#_Toc503787980)

[13 嵌入式linux应用程序移植的步骤 44](#_Toc503787981)

[14 嵌入式linux系统中移植SSH 44](#_Toc503787982)

文档编写目的

本文档主要描述下一代基站承载子系统组网模块的移植方案和调试记录，用以指导相关设计人员快速了解gre、vpn等组网协议，以及代码维护的人员的快速阅读代码，同时也便于开发者自己遗忘时快速查阅。

# 承载子系统主要模块简介

## TCP/IP协议栈支持

实现IP、ARP、TCP、UDP、ICMP、IGMP的功能，由内核实现。

## 时间管理功能块（ntp）

时间管理模块将驻留在BSC上，各BSC通过时间管理模块与时间服务器进行时间同步。

## 语音承载功能块（VoiceBear）

语音承载功能将解决语音在BSC和BSR间、BSC间、BSC和调度子系统间的传输问题。BSR和BSC将采用单播UDP方式传输，BSC间、BSC和调度子系统间将采用IP组播RTP方式传输。

## gre tunnel功能块（tunnel）

对于一些未知的网络，组播数据和单播数据可能无法通过，通过建立gre隧道可以保证组播语音数据和某些单播数据可以通过未知网络。

## vpn功能块（l2tp、ppp）

对于无法直接接入私有网络的偏远基站，可以通过vpn，利用公网来架设接入私网的线路。即实现一个翻墙软件的客户端。

## 动态路由功能块（ospf）

一种路由协议，通过路由器之间通告网络接口状态来建立链路状态数据库，生成最短路径树，维护一张路由表。

## 与B1接口模块（B1AP）

实现与PDC的B1模块交互的接口层，主要包括SNDCP PDU的编解码函数实现。

## 与PDC交互的模块（gre）

这里取名gre和gre tunnel容易混淆，这里虽然使用标准gre协议，但是封装gre协议的ip头是基站和pdc之间自定义的一种私有格式。即基站和pdc交互的ip数据包是非标准的。

### 基站侧实现如下

基站负责维护两张空口至网口转发的通道表

一张以gre报文的key为索引,用以确定pdc->bs数据包所属的通道；

一张以snAPI和ssi为索引，用以确定bs->pdc的数据包所属的通道。

#### 下行：pdc->基站(从pdc接收)->ms

在线程接收处理函数InitGreRecvTask中，接收来自PDC的gre数据包，解析gre头，获取key，通过API调用将数据和key给B1侧。

基站B1通过注册回调函数recvGreFunc来接收来自基站承载侧的gre数据。

#### 上行：ms->基站(向pdc发送)->pdc

B1收到来自空口的gre数据，通过调用sendGreFunc（平台提供，封装gre头，封ip头，用raw socket 发送ip数据包）发给pdc。

### PDC侧实现如下

#### 下行：WAP->eth1->PDC(向基站发送)->eth2->BS->MS

Eth1接收来自路由器的ip数据包，路由至虚拟网络设备，由通道参数将数据,tcp

/ip压缩，封装sndcp头，封装gre头、IP头，然后交由网络协议栈处理。

InitPdc 中会创建Pdc发送线程 PdtPdcTransKthread。

注册的网络设备net\_device中有个字段net\_device\_ops,这个字段中有数据发送函数pdcChannelXmit->pdcprotoDeal->GetSkb2Fransmit->wakeupPdcThread 来唤醒发送线程，读取发送队列。

发送线程唤醒后，SendGTransLinkSkb 发送SKB数据。

发送过程：

DequeueGTransQueueSkb 从全局发送队列获取数据节点

BuildGre 组建gre和sndcp头，CompressProtoHdr 压缩协议头

BuildIpHdrAndSend 组建ip头并发送。（NF\_HOOK dst\_output）

NF\_HOOK(PF\_INET, NF\_IP\_LOCAL\_OUT)(来自本地进程的数据包)🡪 dst\_output🡪 skb->dst->output()—>ip\_mc\_output()/ip\_output()🡪ip\_finish\_output()🡪

NF\_HOOK(PF\_INET, NF\_IP\_POST\_ROUTING)(即将到达电缆的数据包)--> ip\_finish\_output2()-->dev\_queue\_xmit()(发给网卡驱动)

#### 上行：MS->BS->eth2->PDC(从基站接收)->eth1->WAP

Eth2接收来自基站的gre数据，解析gre和sndcp头后，解压tcp/IP头，确定数据包所属的虚拟通道，最后将数据包给虚拟网络设备，进入网络协议栈处理，转发至路由。

GreRcv –> netif\_rx -> Net\_rx\_action🡪Process\_backlog🡪netif\_receive\_skb->ip\_recv

转给ip层 ->NF\_HOOK(PF\_INET, NF\_IP\_PRE\_ROUTING)-> dst\_input()

发给wap服务器。

## 小结

以上各个模块，tcp/ip由内核实现，ntp、语音承载、B1AP、gre的代码实现是跨平台的，所以需要移植的模块有gre tunnel、vpn、ospf三个模块。

# 承载数据发送流程

## 基站B1向交换发送数据的接口

SendB1ByIcp(EADDR\_T chanId, void \*pData , int length)



## 基站BSC向备板或BSR发送数据的接口

SendMsgByIcp(EADDR\_T chanId, MSG\_NODE \*pData)



这两个函数功能差不多，二者均调用SendMsg，以ICP\_SEND 为code将消息发给ICP模块。然后由ICP模块，调用SendFrame 发出去。

## BSC通用消息发送接口



SendMsg—>subSendMsg：

MID相同(同一块板子)，将消息放到本地相应消息客户端(PutMsgToBox)，唤醒本地消息客户端(clientTaskSem)来接收数据.

MID不同，单播发送，SendMsgByIcp🡪SendMsg 发给当前板子的ICP模块.

MID不同，广播发送，向每个通道发送SendMsgByIcp🡪SendMsg发给当前板子的ICP模块.

## ICP模块报文监听模块(SocketListenThread)



ICP通道创建后，由ICP模块以select方式监听。

监听到MSG报文 icpMsgSocketId 以ICP\_RECV消息code发给ICP模块。

监听到B1报文 icpB1SocketId 以ICP\_RECV\_B1消息code发给ICP模块。

ActiveMaster中对socket绑定端口和源地址

监听到B2报文 icpB2SocketId 以ICP\_RECV\_B2消息code发给ICP模块。

监听到B1PDCP报文 icpB1pdcpSocketId 以ICP\_RECV\_B1\_PDCP消息code发给ICP模块。

监听到B1PDCP2报文 icpB2pdcpSocketId 以ICP\_RECV\_B2\_PDCP消息code发给ICP模块。

## ICP本地消息客户端入口 icpFunc



1.ICP\_SEND:调用DoIcpTxJob，SendFrame 将数据发送出去。

2.ICP\_RECV:ReadFrameExt,

读取成功,RecvIcp(icpMsgSocketId,revIcpBuf,ip,CHAN\_TYPE\_MSG)

🡪icpTrunkRxRtnTbl[type] 指针函数。

🡪RecvMsgByIcp

🡪HandoverMsgByIcp

🡪RecvMsg(MID相同，将消息放到本地相应客户端，PutMsgToBox，唤醒本地客户端 clientTaskSem；MID不同，单播发送，SendMsgByIcp-->SendMsg 发给当前板子的ICP模块)

3.ICP\_RECV\_B1:ReadFrame,

读取成功，RecvIcp(icpB1SocketId,revIcpBuf,ip,CHAN\_TYPE\_B1)

🡪icpTrunkRxRtnTbl[type] 指针函数

🡪RecvB1ByIcp

🡪HandoverB1ByIcp

🡪调用由B1注册（RegIcpReportCallBack(ReportICPState, ReportICPData)）的回调函数，将数据发给B1模块。

## 基站各类ICP通道监听的端口

系统启动时，ICP模块初始化函数InitIcpLib中，会创建几类socket，绑定相应端口。

ICP\_MSG\_PORT 16666 bsc与bsr，主备板之间

ICP\_B1\_PORT 17777 bsc与交换

ICP\_B1\_PDCP\_PORT 18888 bsc 与pdc

B1在基站成功注册后，建立与交换之间的ICP B1通道。

OpenIcpChanB1(ecnAddr)

Bps在备板模块初始化中注册远端消息客户端的时候，建立与备板之间的 ICP MSG通道。

OpenIcpChanMsg(peerIp)

## 小结

综上：承载子系统数据传输部分主要关心的是在icp通道中传输的消息如何在tcp/ip协议栈中传输。主要关注以下几个函数。

SendFrame 发送data（sendto）

ReadFrameExt读取来自MSG通道的data （recvfrom）

ReadFrame 读取来自B1通道的data

发送数据、接收数据、接收消息

sendto(id, buf, len, 0, (struct sockaddr \*)&remoteAddr, sockAddrSize)

recvfrom(id, buf, len, 0, (struct sockaddr \*)&remoteAddr, (socklen\_t \*)&sockAddrSize)

recvmsg(bscCtl.rtpRxSockFd, &msg, 0)

# 网络中一帧数据包格式



# 数据结构

## IP头

Ip头

|  |  |  |  |
| --- | --- | --- | --- |
| 字段内容 | 长度 | 值 | 说明 |
| ver\_len | u8 | 0100(ipv4) | 前4bit，版本号；后4bit（ip头长度bit/32），IP包头长度x\*4，20-60字节。 |
| tos | u8 |  | 服务类型，包的优先级 |
| total\_length | u16 |  | Ip包总长度 |
| ident | u16 |  | 与下个字段联合使用，路由器将一个包拆分后，所有拆开的小包标记相同的值 |
| frag\_off\_flags | u16 |  | Flag：3bit，DF是否可以分段，MF分段时，拆分小包除了最后一个包外这个字段置1；frag offset：13bit。片偏移，ip包在分片包中的位置，接收端以此来组装还原IP包。 |
| ttl | u8 |  | 生存时间，Ip包发送时对此字段赋值，每经过一个路由器此值减一，当ttl减为0时，丢弃。防止ip包因为环路而不停转发。 |
| proto | u8 |  | 标识了上层所用的协议，如icmp,tcp,udp,OSPF,gre |
| checksum | u16 |  | 头部校验和，每个路由器会对经过的ip包重新计算校验和。 |
| src | u32 |  | 源地址 |
| dest | u32 |  | 目标地址 |

注：IP头长度20-60个字节。

## Gre头

gre头

|  |  |  |  |
| --- | --- | --- | --- |
| 字段内容 | 长度 | 值 | 说明 |
| C | 1bit |  | 校验和标识位，1，则校验和、偏离4字节必须出现在gre头中，否则不必出现。 |
| R | 1bit |  | 路由标志位，1，则校验和、偏离、路由8字节必须出现在gre头中，否则不必出现。 |
| K | 1bit |  | 密钥标识位，配置为1，则密钥4字节必须出现在gre头中，否则不必出现。 |
| S | 1bit |  | 序列号同步标志位，配置为1，则序列号4字节必须出现在gre头中，否则不必出现。 |
| s | 1bit |  | 严格源路由标志位，通常置0. |
| 递归控制 | 3bit |  | 一般置0 |
| 标志位 | 5bit |  | 一般置0 |
| verson | 3bit |  | 一般置0 |
| protocol | 16bit |  | 若封装ip包，则此字段填0x800 |
| 校验和 | 16bit |  | 可选 |
| 偏离 | 16bit |  | 可选 |
| 密钥 | 32bit |  | 可选 |
| 序列号 | 32bit |  | 可选 |
| 路由 | 32bit |  | 可选 |

注：前4个字节必选，后面5-20个字节可选，

## PPP帧

Ppp帧结构

|  |  |  |  |
| --- | --- | --- | --- |
| 字段内容 | 长度 | 值 | 说明 |
| F | 1B | 7E | 标志 |
| A | 1B | FF | 地址域，无意义 |
| C | 1B | 03 | 控制域，无意义 |
| Protocol | 2B |  | 区分ppp帧的信息域中所承载的数据报的内容。规定协议域所填充的内容必须为奇数，高字节的最低位必须为0，低字节的最低位必须为1.常用0xC021、0x8021、0x0021，分别封装LCP、NCP、IP报文。 |
| 信息域 |  |  | Ip数据报，1500bit |
| FCS | 2B |  | CRC校验 |
| F | 1B | 7E | 标志 |

注：每个ppp数据帧均以0x7E开始和结束。

## L2tp头

L2TP头格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字段内容 | 长度 | 值 | 说明 |
| T | 1bit |  | Type，标识消息类型，0是数据消息，1是控制消息 |
| L | 1bit |  | 置1表示Length(opt)是存在的，对于控制消息，此域必须置1. |
| X | 2bit | 0 | 保留位 |
| S | 1bit |  | Sequence，置1时说明Ns、Nr是存在的，对于控制消息，此域必须置1 |
| X | 1bit | 0 | 保留位 |
| O | 1bit |  | Offset，置1时，offset域是存在的，对于控制消息必须置0 |
| P | 1bit |  | Priority，只用于数据消息，控制消息此域置0，数据消息此位置1表示在传输时应优先处理。 |
| X | 4bit | 0 | 保留位 |
| Ver | 4bit | 2 | 表示l2tp数据报头版本，表示L2tpv2 |
| Length | 16bit | (opt) | 标识整个报文长度，单位字节，控制消息此域一定存在。 |
| Tunnel ID | 16bit |  | 标识l2tp控制连接，只有本地意义，包头中tunnel ID 是接收方的tunnel ID。对于hello控制消息，具有全局性，tunnel id必须为0. |
| Session ID | 16bit |  | 标识隧道中一个会话，只有本地意义，会话两端ID可能不同 |
| Ns | 16bit | (opt) | 表示发送数据消息或控制消息的序号，0开始，以1递增，到216时再从0开始。**当前消息的序号**。 |
| Nr | 16bit | (opt) | 表示下一个期望收到的控制消息，**用于控制消息**，Nr的值设置成上一个接收到的控制消息的Ns+1，这样是对上一个接收到的控制消息的确认。**希望接收的下一个控制消息的序号。** |
| Offset | 16bit | (opt) | Offset size，若存在，表示有效载荷数据的偏移，即载荷数据开始的位置。 |
| Offset Padding | 16bit | (opt) | 填充位 |

## UDP头

UDP头格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字段内容 | 长度 | 值 | 说明 |
| 源端口 | 16bit |  | 一个大于1024的16位数字，由用户程序随机选择。 |
| 目标端口 | 16bit |  | 置1表示Length(opt)是存在的，对于控制消息，此域必须置1. |
| 数据包长度 | 16bit |  | 表示包括udp头在内的udp包长度。即8+数据包的长度。 |
| 校验值 | 16bit |  | 整个UDP报文头和UDP所带的数据的校验和。 |

注：UDP头长度为8字节。

# 一个ip数据包如何在tcp/ip协议栈中传输



# UDP编程步骤

UDP编程的服务器端一般步骤是：

1、创建一个socket，用函数socket()；

2、设置socket属性，用函数setsockopt();可选

3、绑定IP地址、端口等信息到socket上，用函数bind();

4、循环接收数据，用函数recvfrom();

5、关闭网络连接；

UDP编程的客户端一般步骤是：

1、创建一个socket，用函数socket()；

2、设置socket属性，用函数setsockopt();\* 可选

3、绑定IP地址、端口等信息到socket上，用函数bind();可选

4、设置对方的IP地址和端口等属性;

5、发送数据，用函数sendto();

6、关闭网络连接；

跟TCP相比，无需三次握手建立可靠连接之后，收发双方才可以发送、接收数据。

只需要创建socket，绑定IP和端口，就可以发送或接收数据。

# gre组网

## Gre协议概要

本质：通过gre协议，实现一个IP包对另一个IP包的封装。

IP数据穿越未知网络：

在网络中某些设备可能不支持组播，甚至不支持某些单播数据，通过设置gre隧道，隧道两侧可以看做是点对点链路，中间未知网络对隧道两端的设备而言完全是透明的。

原理：数据源头，将私有IP数据包用gre协议封装，再封装为公有IP数据包发送出去，沿途各个设备，即使不支持内部ip地址的单播或组播数据，但是对于这些设备来说，封装过后的外部ip包已是单播数据。

私网数据穿越公网：

GRE，通用路由封装技术。这种技术是在IP数据包的外面再加上一个IP头。通俗的说，就是把私有数据进行一下伪装，加上一个“外套”，传送到其他地方。因为企业私有网络的IP地址通常是自己规划，无法和外部互联网进行正确的路由。而在企业网络的出口，通常会有一个互联网唯一的IP地址。这个地址可以在互联网中唯一识别出来。GRE就是把目的IP地址和源地址为企业内部地址的数据报文进行封装，加上一个目的地址为远端机构互联网出口的IP地址，源地址为本地互联网出口的IP地址的IP头，从而经过通过互联网进行正确的传输。

利弊：适用于建立点对点的链路，但是缺乏多点连线建立链路的能力。

## gre组网模型



## 数据发送和接收流程





## 如何配置隧道

在远程路由器之间配置GRE，总共分为三步:

1．创建虚拟链路（隧道）接口，号码任意，两端可不相同。

2．配置虚拟链路（隧道）接口地址，该地址是在GRE包头中被封装的地址。

3．定义虚拟链路（隧道）的源和目的，因为数据包最终要在公网中传递，所以该地址就是在公网中指导路由器转发数据包的可路由公网IP，也是建立隧道两端路由器的真实公网IP。

比如：基站ecnip：172.168.2.1，dataip：172.168.1.1，交换10.8.52.135

则发送一帧数据包中，

内部ip：172.168.1.1->224.0.0.5，外部IP：10.8.52.158->10.8.52.10

收到一帧数据包中，

外部IP：10.8.52.10->10.8.52.158，内部IP：10.8.52.135->172.168.1.1或172.168.2.1

注意：发的时候内部IP源地址是dataip。收的时候内部目标地址是ecnip。

### 红帽子与路由器之间配置模型

红帽子配置：

ip tunnel add gre1 mode gre remote 10.8.52.10 local 10.8.52.158 ttl 255

ip link set gre1 up

ip addr add 172.168.1.1 peer 172.168.1.2 dev gre1

ip link set gre1 down

ip tunnel del gre1

思科路由器配置：

interface Tunnel0

description to BSC\_M \_GRE

ip address 172.168.1.2 255.255.255.0

ip mtu 1472

ip pim sparse-mode

ip ospf network broadcast

ip ospf mtu-ignore

tunnel source 10.8.52.10

tunnel destination 10.8.52.158

### 开发板与路由器之间配置模型

查看内核是否加载gre模块

# lsmod | grep gre

载入指定模块

# modprobe或insmod gre.ko

# insmod ip\_tunnel.ko

# insmod ip\_gre.ko

开发板配置：

ip tunnel add gre1 mode gre remote 10.8.52.10 local 10.8.56.141 ttl 255

ip addr add 172.168.4.1 peer 172.168.4.2 dev gre1

ip link set gre1 up

ip route add default via 10.8.52.10 dev eth1

路由器配置：

interface Tunnel141

ip address 172.168.4.2 255.255.255.0

ip mtu 1472

ip pim sparse-mode

ip ospf network broadcast

ip ospf hello-interval 1

ip ospf retransmit-interval 1

ip ospf mtu-ignore

tunnel source 10.8.52.10

tunnel destination 10.8.56.141

### 两台主机之间通过2个路由器实现互联模型

host A :  121.207.22.123

host B: 111.2.33.28

在host A上面：

# ip tunnel add gre0 mode gre remote 111.2.33.28 local 121.207.22.123 ttl 255

# ip link set gre0 up

# ip addr add 10.10.10.1 peer 10.10.10.2 dev gre0

创建一个GRE类型隧道设备gre0, 并设置对端IP为111.2.33.28。隧道数据包将被从121.207.22.123也就是本地IP地址发起，其TTL字段被设置为255。隧道设备分配的IP地址为10.10.10.1，掩码为255.255.255.0。

在host B上面：

# ip tunnel add gre0 mode gre remote 121.207.22.123 local 111.2.33.28 ttl 255

# ip link set gre0 up

# ip addr add 10.10.10.2 peer 10.10.10.1 dev gre0

此时，host A 和 host B 建立起GRE隧道了。

检测连通性

# ping 10.10.10.2 (host A)

PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.

64 bytes from 10.10.10.2: icmp\_req=1 ttl=64 time=0.319 ms

64 bytes from 10.10.10.2: icmp\_req=2 ttl=64 time=0.296 ms

64 bytes from 10.10.10.2: icmp\_req=3 ttl=64 time=0.287 ms

撤销GRE隧道

# ip link set gre0 down

# ip tunnel del gre1

ip tunnel del gre1

添加隧道

ip tunnel add gre0 mode gre remote 172.19.20.21 local 172.16.17.18 ttl 255

启动隧道

ip link set gre0 up

给隧道添加IP

ip addr add 10.0.1.1 dev gre0

给隧道添加路由

ip route add 10.0.2.0/24 dev gre0

用户态操作方法可参考iproute2-4.9.0.tar.gz或nettools。

## 移植方法

内核态实现：

提取内核源码\linux\kernel\linux\net\ipv4 ip\_tunnel.c gre.c ip\_gre.c，编译成三个.ko。

在3.10.20内核中，上述三个源文件分别编译成三个模块，最后一个依赖于前二者。内核主要实现了2个功能，一个是网络设备的创建、操作方法，一个是gre协议的实现。

用户态实现：

根据源码iproute2-4.9.0.tar.gz中实现的下列命令，给应用层提供操作网络设备的接口。

ip tunnel add gre0 mode gre remote 10.8.52.10 local 10.8.52.158 ttl 255

ip link set gre0 up

ip addr add 172.168.1.1 peer 172.168.1.2 dev gre0、

ip link set gre0 down

ip tunnel del gre1

GRE隧道配置注意事项

（1）隧道两端的逻辑IP地址要配置正确，这两个地址需要在同一个网段。

（2）隧道两端的物理IP地址要配置正确。

（3）隧道两端的gre key和gre checksum要配置一致。

Gre内核代码路径：

[root@10.8.59.30](mailto:root@10.8.59.30)：/home/wuh/ko/ip\_gretest

密码：1357911

注意：编译内核模块时，使用的内核源码树必须与编译内核使用的内核源码树相同。具体可参见[root@10.8.59.30](mailto:root@10.8.59.30)：/home/wuh/ko/ip\_gretest目录下的Makefile

内核源码树路径：

KERNELDIR ?= /home/fanggj/fanggj-OCTEON-SDK/linux/kernel/linux

## 真实环境调试gre+ospf

### 环境说明

基站gre1口地址：

Dataip：2.1.55.181

Ecnip： 2.2.55.181

Oamip：2.2.55.181

基站外网口eth0的地址：44.58.99.181

对端路由器物理端口1的地址：44.58.99.254（跟基站相连）

对端路由器物理端口2的地址：33.58.99.254（跟交换相连）

交换中心地址：33.58.99.200

隧道两端的物理地址：44.58.99.181、44.58.99.254

隧道两端的虚拟地址：2.1.55.181、2.1.55.254

### 基站配置

:set-net-type:type=3;

:set-bsc-dataip:ipaddr="2.1.55.181";

:set-bsc-ecnip:ipaddr="2.2.55.181";

:set-bsc-oamip:ipaddr="2.2.55.181";

//以下两条配置隧道两端的物理地址及隧道参数

:set-tunnel-outip:ipaddr="44.58.99.181";

:set-tunnel-cfg:c=0 ,k=0 ,key= 0,ipDst="44.58.99.254" ;

//配置一条到交换的静态路由，即下一跳

:set-route-cfg:

unit= 0, enable=1 ,dstip="33.58.99.0" ,mask="255.255.255.0",nexthop= "2.1.55.181";

//配置动态路由ospf

:set-ospf-enable:enable= 1;

:set-ospf-areaid:areaid="192.168.0.1" ;

:set-ospf-helloInterval:interval=10 ;

:set-ospf-DeadInterval:interval= 40;

:set-ospf-priority:priority= 0;

:set-ospf-nettype:type= 2;//广播

### 思科路由器配置

！开启组播

ip multicast-routing

ip multicast multipath

！配置隧道接口

interface Tunnel181

description Connect\_to\_BSEY

ip address 2.1.55.254 255.255.255.0

ip mtu 1472

ip pim sparse-mode

ip ospf network broadcast

ip ospf mtu-ignore

ip ospf hello-interval 10

ip ospf dead-interval 40

tunnel source 44.58.99.254

tunnel destination 44.58.99.181

！配置ospf

router ospf 254

router-id 33.58.99.254

log-adjacency-changes

timers throttle spf 1 1 1

network 2.1.55.0 0.0.0.255 area 192.168.0.1

network 33.58.99.0 0.0.0.255 area 192.168.0.1

！配置路由

**ip route 2.2.55.0 255.255.255.0 2.1.55.181**

注意：一旦基站ecnip和dataip不同，且没有开启ospf，路由器上需要增加一条路由：要到达基站所属交换中心所在网络，下一跳是隧道虚拟口。

### 调试说明

前提：隧道两端的物理地址能ping通

操作：

1.隧道两端的虚拟地址互相ping，能通说明隧道ok；

2.关闭ospf，开启静态路由，基站上ping交换中心 33.58.99.200，能通。

3.关闭静态路由，开启ospf，交换侧路由器上广播33.58.99.0的网络，基站上ping交换中心 33.58.99.200，能通。

5.交换上ping2.1.55.181能通。

6.交换上ping2.2.55.181能通。

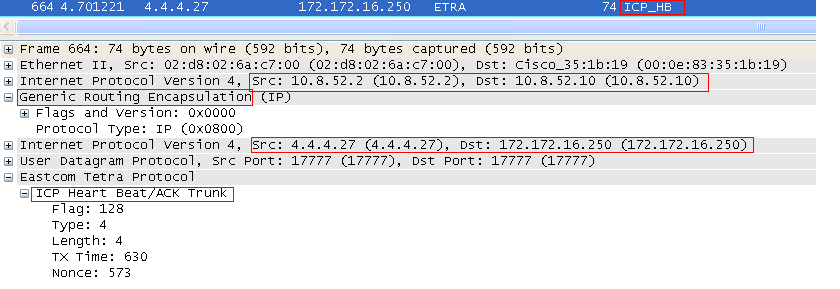
### Wireshark抓包实例

data：3.3.3.27 oam、bscecnip： 4.4.4.27

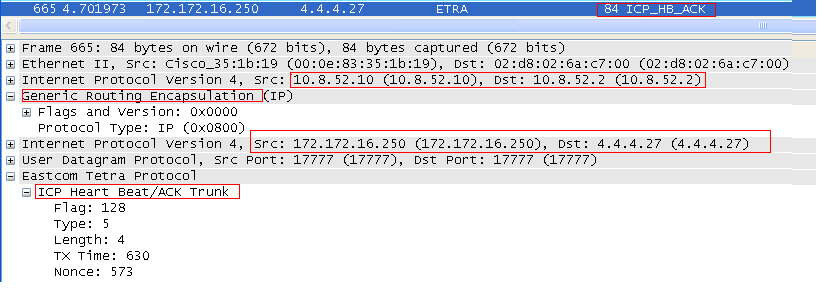
out： 10.8.52.2

ecn: 172.172.16.250

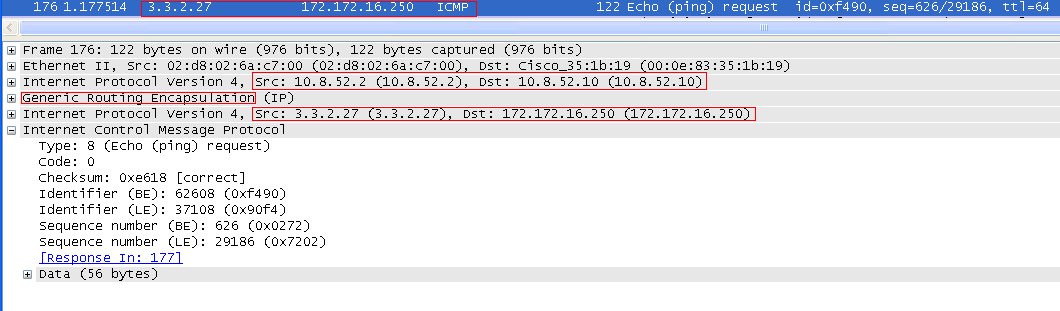
#### 基站发往交换的HB报文：



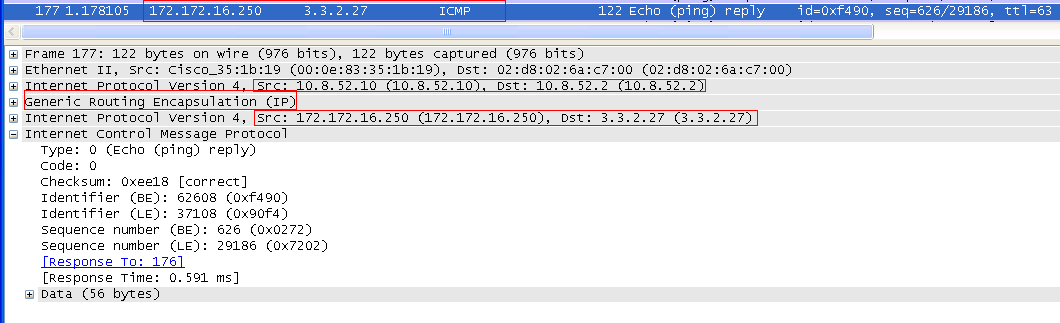
#### 基站收到来自交换的HB\_ACK报文：



#### 基站到交换的ping包



#### 交换到基站的ping响应



可见，

Icp通道中传输的数据，内部Ip头是bsc的ecnip地址；

周期ping交换的ping包的内部ip头是dataip地址。

注意此时ICP两端的一对地址是：4.4.4.27和172.172.16.250

# L2TP组网

## 概要

VPN 属于远程访问技术，简单的说就是利用公用网络架设专用网络。

从原理上来说，VPN就是利用公用网络（通常是互联网）把远程站点或用户连接到一起的专用网络。与使用实际的专用连接（例如租用线路）不同，VPN使用的是通过互联网路由的“虚拟”连接，把公司的专用网络同远程站点或员工连接到一起。公司专用网络一侧成为服务器（LNS），用户一侧称为客户端（LAC）。我们基站实现的就是VPN客户端的功能。

实现VPN客户端，需要用到的两个协议L2TP协议和PPP协议。

## 两种典型的L2TP隧道模式



由远程拨号用户发起：  
远程系统拨入LAC，由LAC通过Internet向LNS发起建立通道连接请求。拨号用户地址由LNS分配；对远程拨号用户的验证与计费既可由LAC侧的代理完成，也可在LNS侧完成。

直接由LAC客户（指可在本地支持L2TP协议的用户）发起：  
此时LAC客户可直接向LNS发起通道连接请求，无需再经过一个单独的LAC设备。此时，LAC客户地址的分配由LNS来完成。基站实现的是这一种模式。

## Pc直接接入LNS（相当于基站接入LNS的模型）

新网络时代的L2TP，没有单独的LAC，L2TP跨越的网络直接就是互联网。

1.企业总部还是要提供L2TP LNS，对互联网上的出差员工提供接入

2.该LNS最好要有一个固定都IP地址，如60.1.99.140，如果地址是变动的，那么需要实用DDNS和一个固定域名如l2tp.tektalk.org绑定起来，为什么呢，如果没有这个地址或域名，你让出差员工拨入到哪呢，这个地址和域名的作用就相当于原生L2TP VPN中的LAC电话号码15400

这里翻墙软件中提供的就是一个域名，一旦IP被封，可以换一个新的IP。

3.LAC功能（或者叫L2TP客户端）一般集成在出差员工PC上，如Windows XP系统自带L2TP客户端 。这里基站的LAC功能集成在bsc内部。

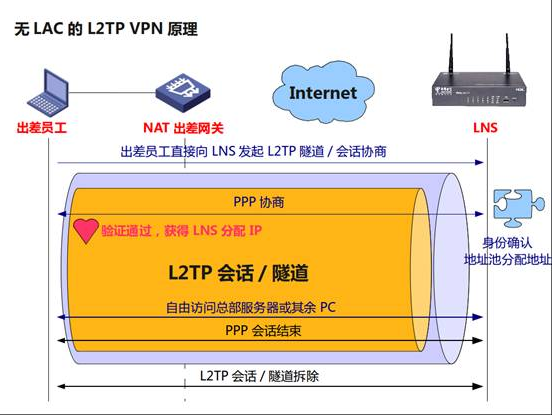
4.出差员工通过设置拨号连接的IP地址或域名，开始向总部LNS发起连接。

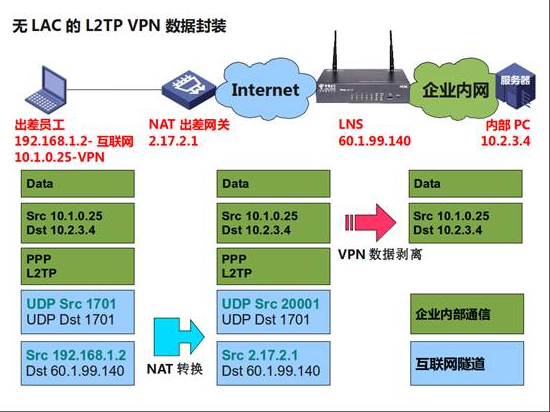
5.建立L2TP隧道后协商L2TP会话 。

6.L2TP会话协商结束，开始进行PPP参数协商，主要是身份确认和地址分配 。

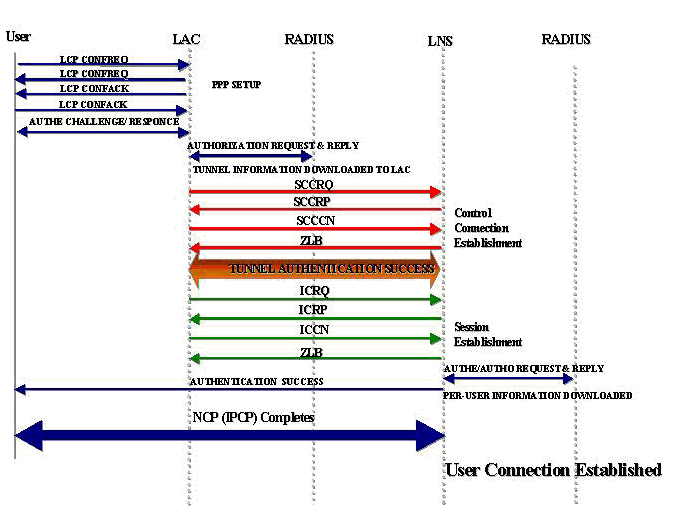
7.获得地址后就算是连接到企业内部网络了，VPN建立成功 。

8.L2TP会话结束，立刻拆除隧道，因为在这种模式中，每个客户端都需要和LNS建立隧道，而这个隧道中只有1个PPP会话





## 客户端和服务器隧道建立的过程





### 控制连接的建立

1.LAC和LNS相互路由可达后，LAC端设置相应的AVP，向LNS端发出SCCRQ报文，请求建立控制连接。

2.LNS收到来自LAC的SCCRQ请求，根据其中的AVP，如果同意建立隧道，便发送SCCRP报文给LAC。

3.LAC收到SCCRP报文进行检查，从中取出隧道信息，并向LNS发送SCCCN报文，表示控制连接建立成功。

4.当消息队列中没有消息的时，LNS发送ZLB消息给对端。

### 会话连接的建立

1.控制连接建立成功之后，一旦检测到用户呼叫，LAC就向LNS发送ICRQ请求在已经建立的tunnel中建立会话。

2.LNS发送ICRP进行回应，表示ICRQ成功，LNS也会在ICRP中标识L2TP session必要的参数。

3.LAC发送ICCN用来回应ICRP，至此L2TP session建立完成。

### 保活

在一个隧道上在指定时间内没有数据和控制消息传输则使用hello报文检测，如果隧道未存活则通道隧道已经失效并重建。

### 会话和隧道的拆除

会话关闭

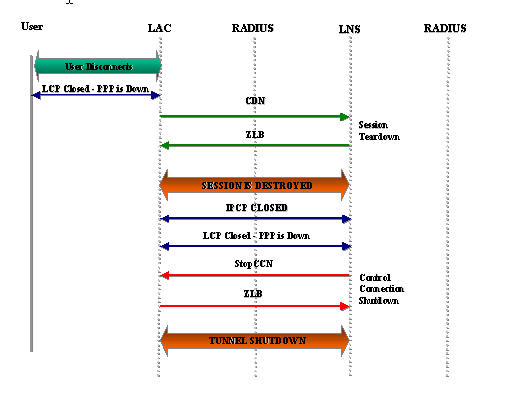
若最后一个会话关闭，则典型情况下控制连接也被关闭。

Lac或lns发起呼叫断开（CDN），对方回应应答（ZLB ACK）

控制连接关闭

任何一端都可发起关闭，关闭消息为Stop CCN。当收到Stop CCN的一方要回

一个ZLB ACK并保持至少一个重传周期以免丢失重发。



## 隧道验证过程

隧道验证是和建立隧道同时进行的，不是单独进行的。

隧道验证过程如下：

1.首先LAC向LNS发SCCRQ请求消息时，产生一个随机的字符串作为本端的CHAP Challenge（SCCRQ携带的字段）发给LNS。

2.LNS收到SCCRQ后，利用SCCRQ携带的CHAP Challenge和本端配置的密码产生一个新的字符串，用MD5算出一个16个字节的Response；同时也产生一个随机的字符串（LNS Challenge），将Response和LNS Challenge放在SCCRP中一起发给LAC。

3.LAC端收到SCCRP后，对LNS进行验证：

利用自己的CHAP Challenge、本端配置的密码、SCCRP，产生一个新的字符串；

用MD5算出一个16字节的字符串；

与LNS端发来的SCCRP中带的LNS CHAP Response做比较，如果相同，则隧道验证通过，否则隧道验证不通过，断掉隧道连接。

4.如果验证通过，LAC将自己的CHAP Response放在SCCCN消息中发给LNS。

5.LNS收到SCCCN消息后，也进行验证：

利用本端的CHAP Challenge、本端配置的密码、SCCCN，得到一个字符串；

然后用MD5算出一个16字节的字符串；

与SCCCN消息中得到的LAC CHAP Response做比较。如果相同，则验证通过，否则拆除隧道。

## PPP协议栈

ppp 位于数据链路层，是一种为同等单元之间传输数据包这样的简单链路设计的链路层协议。

主要负责PPP层数据的封装、压缩与解压缩.另外,它还对普通数据包和Ppp过程的数据包进行了分流,将普通数据包提交到TCP/IP协议栈,而将Ppp过程的数据包放到/dev/ppp设备队列中,等待Pppd去收取并处理.

什么是pppd

pppd是一个用户空间的后台服务进程(daemon)。pppd实现了所有鉴权、压缩/解压和加密/解密等扩展功能的控制协议。pppd只是一个普通的用户进程，pppd与内核中的PPP协议处理模块之间通过设备文件(/dev/ppp)进行通信。pppd有一个辅助工具chat，用来与GSM模组建立会话。它向串口发送AT命令，建立与GSM模组的会话，以便让PPP协议可以在串口上传输数据包。

 pppd是一个后台服务进程(daemon)，是一个用户空间的进程，所以把策略性的内容从内核的PPP协议处理模块移到pppd中是很自然的事了。pppd实现了所有鉴权、压缩/解压和加密/解密等扩展功能的控制协议。pppd只是一个普通的用户进程，它如何扩展PPP协议呢？这就是pppd与内核中的PPP协议处理模块之间约定了，它们之间采用了最传统的内核空间与用户空间之间通信方式：设备文件。设备文件名是/dev/ppp。通过read系统调用，pppd可以读取PPP协议处理模块的数据包，当然，PPP协议处理模块只会把应该由pppd处理的数据包发给pppd。通过write系统调用，pppd可以把要发送的数据包传递给PPP协议处理模块。通过ioctrl系统调用，pppd可以设置PPP协议的参数，可以建立/关闭连接。 在pppd里，每种协议实现都在独立的C文件中，它们通常要实现protent接口，该接口主要用于处理数据包，和fsm\_callbacks接口，该接口主要用于状态机的状态切换。数据包的接收是由main.c: get\_input统一处理的，然后根据协议类型分发到具体的协议实现上。而数据包的发送则是协议实现者根据需要调用output函数完成的。chat是pppd所带一个辅助工具。它和xchat不是一个类型的，xchat用来与人聊天，而chat用来与GSM模组建立会话。它的实现比较简单，它向串口发送AT命令，建立与GSM模组的会话，以便让PPP协议可以在串口上传输数据包。

/dev/ppp

设备文件/dev/ppp。通过read系统调用，pppd可以读取PPP协议处理模块的数据包，当然，PPP协议处理模块只会把应该由pppd处理的数据包发给pppd。通过write系统调用，pppd可以把要发送的数据包传递给PPP协议处理模块。通过ioctrl系统调用，pppd可以设置PPP协议的参数，可以建立/关闭连接。

## L2tp组网模型



## 数据发送和接收流程





控制消息和数据消息的处理

LAC收到ppp报文后进行如下封装：首先为其封装L2TP头，接着封装UDP头，然后封装新的IP头，并从连接的公共网络接口发出去。

L2tp协议本身没有分片功能，但是在进行ip封装时，可以在必要时分片。为了保证报文不分片，封装后的报文大小不能超过实际接口的MTU。

LNS从连接公共网络接口收到该报文后做如下处理：去掉IP头和UDP头，将报文发到L2TP协议模块；l2tp协议去除l2tp头和ppp头，将该报文还原为用户IP报文，并达到私网内部服务器。

## 移植方法

VPN客户端的实现涉及两类消息。

控制消息：用于隧道和会话连接的建立、维护以及传输控制。采用可靠传输，支持对控制消息的流量控制和拥塞控制。由用户态实现。

数据消息：用于封装PPP帧并在隧道上传输。采用不可靠传输，即不重传丢失的数据报文，不支持对数据消息的流量控制和拥塞控制。由内核实现。

内核态：

内核源码如下：\linux\kernel\linux\net\l2tp

主要实现PPP、L2TP的封装，数据包在tcp/ip协议栈中发送接收。

用户态：

主要实现隧道和会话连接的建立、维护以及传输控制。

大致方法：下载xl2tpd的源码，修改配置文件，启动L2TP进程后，调用PPPD进程。

具体过程：

1.下载源码xl2tpd-1.2.4.tar.gz，编译通过。

修改两个配置文件，/etc/xl2tpd/xl2tpd.conf，/etc/ppp/options.l2tpd.client。前者是L2TP隧道、会话建立的配置文件，后者是PPP协商认证的配置文件。

2.用户空间pppd进程的实现。

3.启动l2tp后,调用pppd进程。

红帽子上调试命令：

启动L2TP进程

xl2tpd -D

与LNS建立连接

echo "c myvpn" >/var/run/xl2tpd/l2tp-control

与LNS断开连接

echo "d myvpn" > /var/run/xl2tpd/l2tp-control

成功编译的代码路径如下：

服务器：[root@10.8.60.103](mailto:root@10.8.60.103)

各模块路径：

A:/home/TPS/wuh/testcode/l2tp/xl2tpd-1.2.4

B:/home/TPS/wuh/testcode/l2tp/libpcap-1.3.0

C:/home/TPS/wuh/testcode/l2tp/ppp-2.4.7

编译步骤：

在B目录下，编译生成libpcap.so.1.3.0库。将其拷贝到如下目录

/home/TPS/gaofei/fanggj-OCTEON-SDK/tools-gcc-4.7/mips64-octeon-linux-gnu/lib

在A目录下，编译连接生成可执行程序xl2tpd

在C目录下，编译连接生成可执行程序pppd

Pppd的编译配置

/home/TPS/wuh/testcode/ppp-2.4.5/

/home/TPS/wuh/testcode/l2tp/ppp-2.4.7/

./configure --host=mips64-octeon-linux

make CC=/home/zhengyk/fanggj-OCTEON-SDK/tools-gcc-4.7/bin/mips64-octeon-linux-gnu-gcc

W:\testcode\ppp-2.4.5\pppd Makefile文件

LIBS += -lcrypt -lpthread #wuh add

ppp程序依赖的.ko

ppp驱动：slhc.o ppp\_generic.o pppox.o

是pppd与内核通信的接口。

ppp线路规程：crc-ccitt.o ppp\_async.o

是tcp/ip协议栈与tty驱动通信的接口。

ppp\_synctty.o

Linux中PPP实现主要分成两大部分：PPPD和PPPK。PPPD是用户态应用程序，负责PPP协议的具体配置，如MTU、拨号模式、认证方式、认证所需用户名/密码等。 PPPK指的是PPP内核部分，包括上图中的PPP驱动和PPP线路规程。PPPD通过PPP驱动提供的设备文件接口/dev/ppp来对PPPK进行管理控制，将用户需要的配置策略通过PPPK进行有效地实现，并且PPPD还会负责PPP协议从LCP到PAP/CHAP认证再到IPCP三个阶段协议建立和状态机的维护。因此，从Linux的设计思想来看，PPPD是策略而PPPK是机制；从数据收发流程看，所有控制帧(LCP,PAP/CHAP/EAP,IPCP/IPXCP等)都通过PPPD进行收发协商，而链路建立成功后的数据报文直接通过PPPK进行转发，如果把Linux当做通信平台，PPPD就是Control Plane而PPPK是DataPlane。

开发板上调试：

mkdir /etc/xl2tpd

mkdir /etc/ppp

cp xl2tpd.conf /etc/xl2tpd/

cp options.l2tpd.client /etc/ppp/

chmod 777 /etc/xl2tpd/xl2tpd.conf

chmod 777 /etc/ppp/options.l2tpd.client

mkdir /var/run/xl2tpd

touch /var/run/xl2tpd/l2tp-control

./xl2tpd –D

echo "c BSC1" >/var/run/xl2tpd/l2tp-control

## xl2tpd配置和运行

### Server配置

#cat /etc/xl2tpd/xl2tpd.conf

[global]

listen-addr = 10.8.52.177 #listen-addr 是外网IP；监听哪个网口，全0表示监听所有网口

port = 1701 #监听L2TP协议的端口

auth file =/etc/ppp/chap-secrets

[lns default]

ip range =3.3.3.100-3.3.3.200 #分配给客户端虚拟ip范围

local ip = 3.3.3.254 #local ip 本地虚拟ip

require chap = yes #以下三项表示服务器要求认证

refuse pap = yes

require authentication = yes

name = LinuxVPNserver #服务器的名称

ppp debug = yes #开启ppp的debug模式

pppoptfile =/etc/ppp/options.xl2tpd .server #定义ppp的相关参数

length bit = yes

#ppp的配置文件

options.xl2tpd.server:

#cat /etc/ppp/options.xl2tpd

ipcp-accept-local

ipcp-accept-remote

#ms-dns  8.8.8.8 #是否给客户端的分配dns

Noccp #禁用 CCP（Compression Control Protocol，压缩控制协议）协商

auth

crtscts

idle 1800 #若链路闲置了1800s，则pppd必须断开连接

mtu 1410 #最大传输单元

mru 1410 #最大接收单元

nodefaultroute

debug #启用连接调试工具, 通过相应地配置 /etc/syslog.conf 将该信息定向到一个文件。

lock

proxyarp #用户可以通过 pppd 创建代理地址转换协议条目

connect-delay 5000 #只有在启动了connect和pty选项时，等待期间才适用。

#lcp-echo-interval 30

#每30s向对等方发送 一个LCP Echo-Request 帧

#链路维护阶段发送维护报文的时间间隔，若30\*4s内收不到维护报文，链路自动断开，默认为0，网络不好，则会往复拨号。

#lcp-echo-failure  4

#l2tp的使用的用户名和密码

chap-secrets:

# cat /etc/ppp/chap-secrets

# Secrets for authenticationusing CHAP

# client    server   secret            IP addresses

"prin"       \*    "prin"        \*

"vpn"       \*    "123456"        \*

第一列是用户名，第二列是server的名称即为xl2tpd.conf中的name，参数\*代表所有，第三列是密码，第四列是允许的IP(\*为全部允许)

### Client配置

xl2tpd.conf

# cat /etc/xl2tpd/xl2tpd.conf | grep -v"^;"

[global]

port = 1701

debug avp = yes #下列5句用于调试

debug network = yes

debug packet = yes

debug tunnel = yes

debug state = yes

access control = no

[lac myvpn] #这里myvpn是隧道的名称，拨号和断开连接的时候需要

name = vpn # name？暂时不明，随意配置。

hostname=BSC1 # 客户端的名称，注意，需要跟路由器一致，即set-l2tp-lacname一致。

lns = 10.8.52.177

redial = yes #重播，not sure

redial timeout = 1  #重播间隔，not sure

max redials = 20 #最大重播次数，not sure

refuse pap = yes

require authentication = no

ppp debug = yes

pppoptfile = /etc/ppp/myvpn.options.l2tpd.client

#cat /etc/ppp/myvpn.options.l2tpd.client

myvpn.options.l2tpd.client

remotename Router #指定对等方的名称,比如LinuxVPNserver，不检查

user       "vpn" #设置VPN连接的用户名

password      "123456" #设置VPN连接的密码

unit 0 #将ppp接口的单元号设置为0

lock

nodeflate #禁用deflate压缩

nobsdcomp #禁用bsd压缩

noauth #/etc/ppp/auth-up脚本不会执行，不要求对等方证明自己的身份

persist #在连接终止后不退出，而是尝试重新打开连接。

nopcomp #在接收和传送方向禁用协议字段压缩协商。

noaccomp #在两个方向（发送和接收）都禁用 HDLC 地址/控制压缩

maxfail  5 #在连续 n 次连接尝试失败后终止

### 运行xl2tpd

service xl2tpd start (client and server)

echo "c myvpn" >/var/run/xl2tpd/l2tp-control (client)

几秒钟之后，在/var/log/messages下可以看到l2tp隧道建立、ppp0接口开启的日志输出

### 配置路由

客户端：

ip -6 route add default devppp0

服务器：

Ip -6 route add2001:192:168:3::/64 dev ppp0 （2001:192:168:3::/64是服务器所在的IPv6网段）

### crontab配置

使用crontab来实现ppp接口的例行检查和断线重连。

客户端：（这里是openwrt）

\* \* \* \* \* test -n $( ifconfig | grep ppp0 ) && echo 'c vpn1' >/tmp/run/xl2tpd/l2tp-control && sleep 3 && ip -6 route adddefault dev ppp0

\* \* \* \* \* test -n $( ip -6 route show | grep ppp0 ) && ip -6route add default dev ppp0

服务器：（Centos）

\* \* \* \* \* root test -n $( ip -6 route show | grep ppp0 ) && ip -6route add 2001:192:168:3::/64 dev ppp0

重启，拨号测试，断线重连测试

## 真实环境调试vpn+ospf

### 环境说明

开发板右边第一个是eth0，第二个是串口

基站ppp1口地址：

Dataip：3.3.3.181

Ecnip： 2.2.55.181

Oamip：2.2.55.181

基站外网口eth0的地址：44.58.99.181

对端路由器物理端口1的地址：44.58.99.254（跟基站相连）

对端路由器物理端口2的地址：33.58.99.254（跟交换相连）

交换中心地址：33.58.99.200

隧道两端的物理地址：44.58.99.181、44.58.99.254

隧道两端的虚拟地址：3.3.3.181、3.3.3.254

### 基站配置

:set-net-type:type=2;

//配置隧道两端的物理地址

:set-tunnel-outip:ipaddr="44.58.99.181",mask="255.255.255.0";

:set-l2tp-ipaddr:ipaddr="44.58.99.254";

//配置隧道本端的虚拟地址

:set-bsc-dataip:ipaddr="3.3.3.181",ipmask="255.255.255.0";

注意：这里配置dataip没用，ppp0的IP是ppp进程协商产生的，但是此处mask需要配置，否则ospf邻居关系建立不起来（思科路由器上无法配置掩码全是255的虚拟ip）

//配置一条到交换的静态路由

:set-route-cfg:unit=0,enable=1,dstip="33.58.99.0",mask= "255.255.255.0",nexthop= "3.3.3.181";

:set-l2tp-hellotimeout:interval=10;

//设置拨号用户名和密码

:set-ppp-username:username="wuh";

:set-ppp-pwd:pwd="wuh";

//设置来l2tp客户端的名称

:set-l2tp-lacname:lacname="BSC1";

如此配置基站启动将产生2个配置文件如下：

#1 /etc/ xl2tpd /xl2tpd.conf

[global]

port = 1701

debug avp = yes

debug network = yes

debug tunnel = yes

debug state = yes

access control = no

[lac myvpn]

name =none

hostname =BSC1

lns =44.58.99.254

refuse pap = yes

require authentication = no

ppp debug = yes

pppoptfile = /etc/ppp/options.l2tpd.client

length bit = yes

#2 /etc/ppp/options.l2tpd.client

remotename Router

user wuh

password wuh

unit 0

lock

nodeflate

nobsdcomp

noauth

persist

nopcomp

noaccomp

maxfail 5

### 思科路由器配置

//创建拨号的用户名和密码

username wuh password 0 wuh

//启用VPDN功能

vpdn enable

//建立一个虚拟拨号组1

vpdn-group 1

! Default L2TP VPDN group

accept-dialin //设置允许客户端接入

protocol l2tp //启用l2tp隧道协议

virtual-template 1 //建立一个虚拟接口1

terminate-from hostname BSC1 //设置拨号客户端的名称

l2tp tunnel hello 10

no l2tp tunnel authentication //关闭l2tp隧道认证功能

l2tp tunnel timeout no-session 0

l2tp tunnel retransmit retries 5

l2tp tunnel retransmit timeout max 5

interface Virtual-Template1

ip address 3.3.3.254 255.255.255.0 //配置隧道本端的虚拟地址

ip pim sparse-mode

ip ospf network broadcast

ip ospf mtu-ignore

peer default ip address pool mypool1 //设置获得IP地址对应的地址池

ppp authentication chap ms-chap //设置PPP认证方式chap 或ms-chap

router ospf 254

network 3.3.3.0 0.0.0.255 area 192.168.0.1

ip local pool mypool1 3.3.3.181 //配置L2TP地址池

或者

ip local pool mypool1 3.3.3.100 3.3.3.181 //配置L2TP地址池

### 思科路由器上查看vpn调试日志

debug aaa authentication - 显示 AAA/TACACS+ 身份验证的信息。

debug aaa authorization - 显示有关 AAA/TACACS+ 授权的信息。

debug aaa accounting - 在可记帐事件出现时显示其相关信息。使用此命令显示的信息与用于向服务器传输记帐信息的记帐协议无关。

debug tacacs+ -显示与TACACS+相关的详细调试信息。

debug vtemplate - 显示从虚拟模板克隆虚拟访问接口时到虚拟访问接口因呼叫结束而关闭时虚拟访问接口的克隆信息。

debug vpdn error -显示防止一个PPP隧道被设立造成一个已建隧道被关闭的错误。

debug vpdn events -显示关于正常PPP隧道建立或关闭的一部分事件的消息。

debug vpdn l2x-errors - 显示妨碍建立第二层或妨碍其正常操作的第二层协议错误。

debug vpdn l2x-events - 显示第二层的正常 PPP 隧道建立或关闭过程中的事件相关消息。

开启debug之后，terminal monitor

将在显示器上看到日志输出。

### 思科路由器上查看ospf调试日志

Router#debug ip ospf packet

Router#debug ip ospf events

开启debug后，terminal monitor将在显示器上看到日志输出。

关闭日志：

no debug ip ospf packet

no debug ip ospf events

### Wireshark抓包

抓包命令：

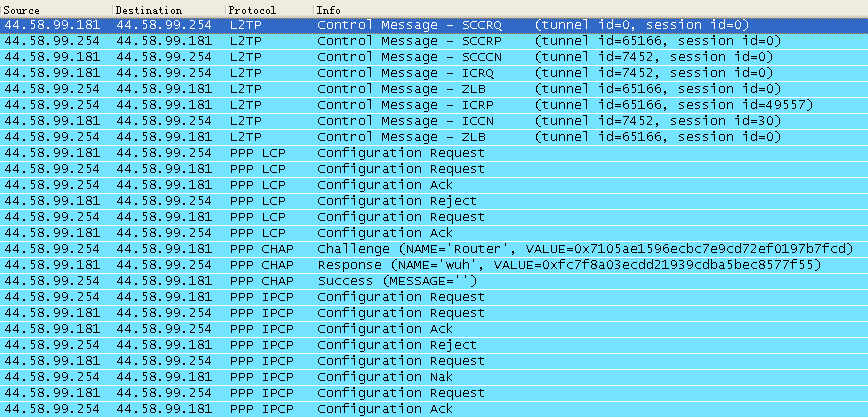
tcpdump -i eth0 –w 123.pcap

tcpdump -i eth0 host 10.8.52.177 –w 123.pcap

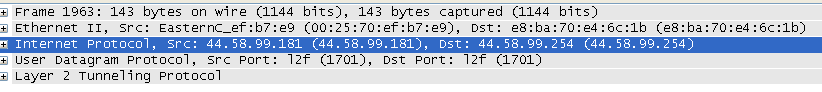
SVN：\基础平台文档\下一代基站承载开发\L2tp\调试日志\vpn环境ping包日志.pcap,

vpn连接断开完整日志.pcap

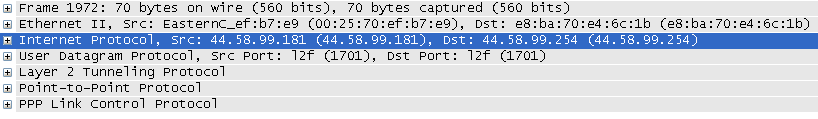
L2TP的控制消息



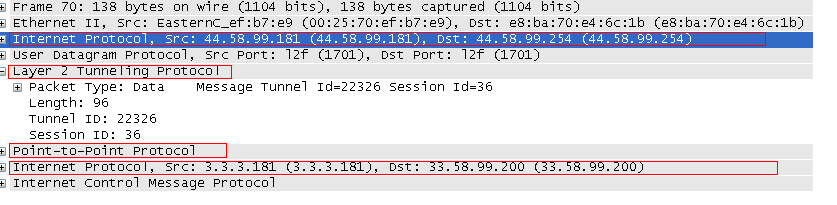
隧道建立的消息

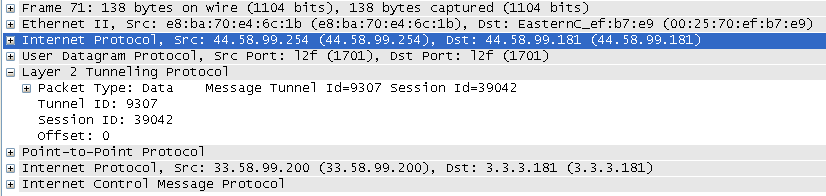


Ppp认证的消息



L2TP的数据消息





# ospf模块

## 代码、编译

开源代码：zebra-0.95a

成功编译的代码路径如下:

服务器：[root@10.8.60.103](mailto:root@10.8.60.103)

/home/TPS/wuh/testcode/platform\_temp/bsc/bear/zebra-0.95a/

编译生成可执行程序zebra和ospfd

正确的交叉编译环境配置

./configure --disable-bgpd --disable-ripd --disable-ripngd --enable-irdp --disable-bgp-announce CC=/home/zhengyk/fanggj-OCTEON-SDK/tools-gcc-4.7/bin/mips64-octeon-linux-gnu-gcc

--host=mips64-octeon-linux

开源代码：quagga-0.99.19（下一代基站使用此份代码）

成功编译路径：

服务器：[root@10.8.60.103](mailto:root@10.8.60.103)

/home/TPS/wuh/testcode/quagga-0.99.19

配置方法：

./configure CC=/home/zhengyk/fanggj-OCTEON-SDK/tools-gcc-4.7/bin/mips64-octeon-linux-gnu-gcc --host=mips64-octeon-linux --enable-user='usrquagga' --enable-group='quagga'

编译方法：

home/TPS/wuh/testcode/quagga-0.99.19路径下执行make，将在本目录下生成下列4个文件

./zebra/.libs/zebra、./ospfd/.libs/ospfd、./lib/.libs/libzebra.so 、./ospfd/.libs/libospf.so

最后，将这4个文件放在打包目录中统一执行打包。

## 配置文件

### #zebra.conf

hostname router

password zebra

!enable password zebra

interface eth0 !note:dataip网口

description Interface to External Network

ip address 44.58.99.181/24 !note:dataip

ipv6 nd suppress-ra

!

log file /usr/local/etc/zebra.log

!

!

line vty

!

### #ospfd.conf

hostname ospfd

password zebra

!enable password zebra

!

interface eth0 !note:dataip网口

ip ospf hello-interval 10

ip ospf dead-interval 40

ip ospf priority 0

ip ospf network broadcast

!

router ospf

ospf router-id 44.58.99.181

ospf abr-type cisco

network 44.58.99.0/24 area 192.168.0.1 !note:dataip所在的网络

!

log file /usr/local/etc/ospfd.log

!

!

# 最终下一代基站中运行的各模块代码路径和编译方法

## gre、ospf、vpn的初始化代码

编到bps库中

W:\wuh\_view\SW\_BPS\BSEY\bsc

/home/TPS/wuh/wuh\_view/SW\_BPS/BSEY/bsc 执行make,生成bps库。

/home/TPS/wuh/wuh\_view/SW\_COMM/lib/SW\_BPS/Image/app/BSEY/ET\_BSEY\_img

执行make app，生成image

搜索"InitVPN",查看vpn相关代码

搜索“startospf” ,查看ospf相关代码

搜索“InitGreTunel” ,查看gre相关代码

## gre模块的内核模块代码

W:\wuh\_view\SW\_BPS\BSEY\bsc\bear\kernel\gre

/home/TPS/wuh/wuh\_view/SW\_BPS/BSEY/bsc/bear/kernel/ gre

执行make，生成gre.ko、 ip\_gre.ko、ip\_tunnel.ko

## Ospf模块的代码

W:\testcode\quagga-0.99.19

/home/TPS/wuh/testcode/quagga-0.99.19

执行make，生成

./zebra/.libs/zebra、./ospfd/.libs/ospfd、./lib/.libs/libzebra.so 、./ospfd/.libs/libospf.so

## vpn模块ppp协议栈内核模块代码

W:\wuh\_view\SW\_BPS\BSEY\bsc\bear\kernel\l2tp

/home/TPS/wuh/wuh\_view/SW\_BPS/BSEY/bsc/bear/kernel/l2tp

执行make

生成slhc.ko、 pppox.ko 、ppp\_generic.ko 、ppp\_asyncN.ko

## vpn模块pppd用户态代码

W:\testcode\ppp-2.4.5\ppp2.4.5-103

/home/TPS/wuh/testcode/ppp-2.4.5

执行

make CC=/home/zhengyk/fanggj-OCTEON-SDK/tools-gcc-4.7/bin/mips64-octeon-linux-gnu-gcc

生成pppd

搜索"wuh"，可以看到我添加的代码

## vpn模块的l2tp协议栈代码

W:\testcode\xl2tpd-1.2.4-191test\si-124

/home/TPS/wuh/testcode/xl2tpd-1.2.4-191test

执行make，生成xl2tpd

## 打包bear\_install.bin

W:\platform\_temp\ET\_BSEY\_img2

/home/TPS/wuh/platform\_temp/ET\_BSEY\_img2 执行make bear

# 打包说明

将

/home/TPS/wuh/platform\_temp/ET\_BSEY\_img2/releasefiles

目录下文件(gre.ko、 ip\_gre.ko、ip\_tunnel.ko、slhc.ko、 pppox.ko 、ppp\_generic.ko 、ppp\_asyncN.ko、 ospfd、ospfd.sh、zebra、zebra.sh、libospf.so、libzebra.so、xl2tpd、pppd、tunnel.sh、sysinfo.sh)和前一个目录的bear\_install\_script.sh脚本打包，打包后文件名为bear\_install.bin (可以在打包的makefile中指定)。

至于ospf的配置文件（zebra.conf 、ospfd.conf），vpn的配置文件（xl2tpd.conf、options.l2tpd.client），无需在此打包，它将在image程序中进行动态构造。

在etra.sh脚本中添加一行执行./ bear\_install.bin。这样上电之后，内核加载成功后，执行etra.sh脚本时，会执行到bear\_install.bin，这样解包后，各个模块会被安装。

# 实际调试中问题

1.gre组网，调试ospf，ospf识别不了gre1口

问题解决：更换ospf的开源代码。使用quagga-0.99.19替换zebra-0.95a。

2.vpn组网隧道建立失败  
原因：基站实现的L2TP客户端没有配置主机名称，路由器需要对主机名称进行检查。

3.vpn组网隧道、会话可以建立，但是ppp认证失败。

原因：基站软件上缺少ppp认证所依赖ppp\_async.ko模块。

4.vpn连接失败，基站重启，串口死掉

重启原因：vpn连接失败，ppp0口没有起来，后续配置静态路由时，操作ppp0口会导致系统重启。

串口原因：image中启动vpn连接，获取信号量以等待ppp进程的启动，等待时间若为20s，vpn连接失败，串口会连接不上；目前将这里等待时间改为5s，没有问题。

5.gre组网，icp创建失败

原因：

icp数据需要绑定源地址（基站的ecnip）。目前代码绑定发生在创建gre隧道之前，由于隧道创建之前，ecnip还没配置到gre口，故此时绑定会失败。

后面创建icp的时候，发送的udp数据包，在组建ip头时，由于没有做地址绑定，故填的是gre口的默认地址（即首个地址，dataip），这个地址交换不认，所以创建icp失败。

目前代码：socket地址绑定在InitRdt()中调用， 网络相关的配置在InitBscMng()中调用， InitRdt()的调用在InitBscMng()之前！

vxworks下，这样的顺序调用不会有问题，但是linux下会有问题。

# 嵌入式linux应用程序移植的步骤

1、准备好交叉编译环境

在安装有Linux操作系统的PC上安装对于平台的交叉编译器，并将交叉编译器加到环境变量中，如export PATH=$PATH:/opt /toolchain/rsdk-1.3.6-5281-EB-2.6.30-0.9.30/bin, 然后在终端界面里看交叉编译器版本，如敲mips-linux-gcc –v，显示版本号则表示安装成功。

2、准备好源代码

准备好需要编译的源代码库包，如capwap-0.93.3，需要注意的是，不仅需要准备要编译的源代码库包，还需要准备该源码包依赖的包，例如capwap-0.93.3依赖安全加密相关的包openssl和多线程相关的包pthread，这些包也需通过编译成静态库或动态库供主承销包调用。

3、修改Makefile

一般的源码库可以通过运行./Configure来制定编译器gcc，目标板的架构已经生产应用程序和库的目录。如果没有Configure文件就需要手动打开Makefile文件来修改，主要需要修改的地方有：(1)编译器的类型，(2)需要库的头文件路径;(3)需要库的链接路径(4)生成应用程序的路径。如openCapwap移植过程，将CC=gcc行用CC=mips-linux-gcc替换。

4、编译源代码

在源码包的主路径下运行Make,除非运气特别好，一般情况下是会报错的，需要根据报错的类型进行相应的修改。常见的报错类型有：(1)依赖的库包不支持该CPU架构，需要更换该架构的库包，如Capwap自带的openssl库不支持 mips。#error "This openssl-devel package does not work your architecture?"(2)依赖的库没有经过交叉编译就拿来用了，如#error“./static/libssl.a: could not read symbols: File in wrong format”即libssl.a库文件格式是X86架构下的不支持mips架构。(3)源代码中有c语言方面的错误，一般是和交叉编译器版本不匹配引起的。

一步步解决完这些错误后，然后终于可以生产对应目标板的应用程序了。但是生产相应的应用程序才是万里长征的第一步，让程序正确的运行才是最终目标。

5、安装应用程序

安装应用程序有两种方法，一是将应用程序放到目标板的文件系统中，通过烧镜像的方法将程序下载到目标板上;另一种是通过像tftp的方法下载到目标板上。后一种方法便捷灵活，在调试程序的过程中应用较多。需要注意的是，还需要将应用程序需要的动态库也下载到目标板上，应用程序才能跑。例如在我们移植capwap中出现，在完成烧录后，运行WTP报错。Error：系统化找不到 pthread.so。分析：在终端中进入lib目录，发现缺少libpthread.so动态库。故原因在于RTL8198目标板SDK编译时没有将 libpthread.so动态库添加到目标板系统的lib库文件当中。

6、运行调试应用程序

调试应用程序让其能够正确的工作，才是移植工作最重要的部分，这需要对应用程序的流程很熟悉，然后通过打印日志的方法看程序运行的路径，分析日志与正确的流程的差异来确定出错的地方。常见的出错地方有：(1)内存分配函数;(2)系统位数不一样;(3)字节顺序问题(大端小端);(4)浮点数的表示问题等。如我们在移植过程中遇到的malloc函数行为不同的问题。以上都是平时移植过程中需要重点注意的地方。

# 嵌入式linux[系统](http://www.2cto.com/os/)中移植SSH

硬件平台：s3c6410（arm）

操作系统：linux

一直用的串口调试机器，不过有线的串口也有不方便之处，调试设备必须抱着笔记本，所以想办法把SSH移植到机器中  www.2cto.com

Dropbear是一个相对较小的SSH服务器和客户端。它运行在一个基于POSIX的各种平台。 Dropbear是开源软件，在麻省理工学院式的许可证。 Dropbear是特别有用的“嵌入”式的[Linux](http://www.2cto.com/os/linux/)（或其他Unix）系统，如无线[路由器](http://www.2cto.com/net/router/)。(一个完整的openssh大小有7M左右)

  www.2cto.com

准备

zlib-1.2.3.tar.gz [点击进入下载地址](http://download.csdn.net/source/3287567)

dropbear-0.52.tar.gz [点击进入下载地址](http://download.csdn.net/source/3287523)

1.交叉编译zlib-1.2.3.tar.gz

解压zlib-1.2.3.tar.gz，创建/usr/local/zlib目录；

./configure --prefix =/usr/local/zlib

修改Makefile

CC=arm-linux-gcc

AR=arm-linux-ar rc

RANLIB=arm-linux-ranlib

make；make install;

2.交叉编译dropbear-0.52.tar.gz

解压dropbear-0.52.tar.gz，创建/usr/local/dropbear；

./configure –prefix=/usr/local/dropbear –with-zlib=/usr/local/zlib/ CC=arm-linux-gcc –host=arm

注：这里要求使用绝对路径-如果希望放在home目录可以使用${HOME}/usr/locale/zlib与${HOME}/usr/locale/dropbear

make；

make scp；

make install；

cp scp /usr/local/dropbear/bin/

3.移植到6410：

将/usr/local/dropbear/bin/和/usr/local/dropbear/sbin/下的文件都复制到板上/usr/sbin目录。

在板上生成server key:

cd /etc

mkdir dropbear

cd dropbear

dropbearkey -t rsa -f dropbear\_rsa\_host\_key

dropbearkey -t dss -f dropbear\_dss\_host\_key

在启动脚本/etc/init.d/rcS中增加: /usr/sbin/dropbear

3.配置ssh：

修改root密码（如果无法修改请在/etc中添加passwd与group文件）

pwsswd文件中如下

root:9.iWjoCN45yCg:0:0:Linux User,,,:/root/:/bin/sh

我这里测试goup文件空的也可以

或者：

root状态下在板上输入命令添加用户michael和设置密码 //最好直接改root密码

adduser michael

注：

如果连接出现以下问题：

[plain]

dragon@dragon-ubuntu10:/srv/nfs/rootfs/usr/local/dropbear/bin$ ssh 10.17.98.2 -l root

root@10.17.98.2's password:

PTY allocation request failed on channel 0

shell request failed on channel 0

解决方法：

在dev中创建pts目录:mkdir /dev/pts -p

在fstab文件中添加一段 none /dev/pts devpts defaults 0 0

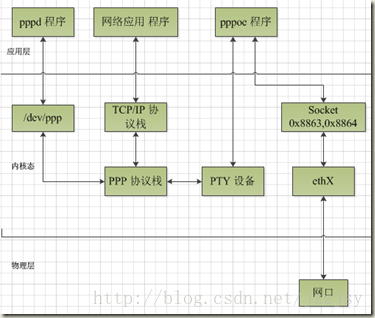
#vi /etc/fstab

none  /dev/pts devpts defaults  0  0

然后 mount -a

问题应该能够得到解决

用xshell连接就能连上去了~ 如果不行的话直接用root用户试试~



如何启用 PPP 调试

成为超级用户或承担等效角色。

创建日志文件以保存 pppd 的输出。

# touch /var/log/pppdebug

在 /etc/syslog.conf 中为 pppd 添加以下 syslog 工具。

daemon.debug;local2.debug /var/log/pppdebug

重新启动 syslogd。

# pkill -HUP -x syslogd

使用以下 pppd 语法，为对特定对等点的呼叫启用调试功能。

# pppd debug call peer-name

peer-name 必须是 /etc/ppp/peers 目录中的某个文件的名称。

查看日志文件的内容。

# tail -f /var/log/pppdebug