第二章 openwrt 源码配置

1. Wifi 开启与配置

控制 wifi 参数的文件是 openwrt/package/mac80211/files/lib/wifi/mac80211.sh，打开这个文件，注释或者删除掉

#REMOVE THIS LINE TO ENABLE WIFI：

option disabled 1

上面这一行

```
config wifi-device  radio$devidx
        option type     mac80211
        option channel  ${channel}
        option hwmode   11${mode_11n}${mode_band}
$dev_id
$ht_capab
        # REMOVE THIS LINE TO ENABLE WIFI:

config wifi-iface
        option device   radio$devidx
        option network  lan
        option mode     ap
        option ssid     UAV
        option encryption none

EOF
        devidx=$(($devidx + 1))
        done
}
```

通过该文件，可以更改模式，ssid， 加密等等参数，这些都会作为编译完成后系统的初始值存在。

2.ip 网关的设置

Openwrt 在启动的时候，会通过运行 uci-defaults.sh 这个脚本程序来设置 IP 等基本参数

该脚本程序在/openwrt/package/base-files/files/lib/functions/下

打开该脚本程序

```
ucidef_set_interface_lan() {
        local ifname=$1

        uci batch <<EOF
set network.lan='interface'
set network.lan.ifname='$ifname'
set network.lan.type='bridge'
set network.lan.proto='static'
set network.lan.ipaddr='192.168.1.1'
set network.lan.netmask='255.255.255.0'
EOF
}
```

该函数便是设置 Lan 口的网络参数

```
ucidef_set_interface_wan() {
        local ifname=$1

        uci batch <<EOF
set network.wan='interface'
set network.wan.ifname='$ifname'
set network.wan.proto='dhcp'
EOF
}
```

该函数设置 wan 口参数

3.发送自定义数据包

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>          // close()
#include <string.h>          // strcpy, memset(), and memcpy()
#include <netdb.h>           // struct addrinfo
#include <sys/types.h>       // needed for socket(), uint8_t, uint16_t, uint32_t
#include <sys/socket.h>      // needed for socket()
#include <netinet/in.h>      // IPPROTO_ICMP, INET_ADDRSTRLEN
#include <netinet/ip.h>      // struct ip and IP_MAXPACKET (which is 65535)
#include <netinet/ip_icmp.h>  // struct icmp, ICMP_ECHO
#include <arpa/inet.h>       // inet_pton() and inet_ntop()
#include <sys/ioctl.h>       // macro ioctl is defined
#include <bits/ioctls.h>     // defines values for argument "request" of ioctl.
#include <net/if.h>          // struct ifreq
#include <linux/if_ether.h>   // ETH_P_IP = 0x0800, ETH_P_IPV6 = 0x86DD
#include <linux/if_packet.h>  // struct sockaddr_ll (see man 7 packet)
#include <net/ethernet.h>
```

```
#include <errno.h>            // errno, perror()
#define ETH_P_DEAN 0x8874 //自定义的以太网协议 type

int main (int argc, char **argv)
{
   int i, datalen,frame_length, sd, bytes;
   char *interface="eth1";;
   uint8_t data[IP_MAXPACKET];
   uint8_t src_mac[6];
   uint8_t dst_mac[6];;
   uint8_t ether_frame[IP_MAXPACKET];
   struct sockaddr_ll device;
   struct ifreq ifr;

   // Submit request for a socket descriptor to look up interface.
   if ((sd = socket (PF_PACKET, SOCK_RAW, htons (ETH_P_ALL))) < 0) {//第一次创建
socket 是为了获取本地网卡信息
      perror ("socket() failed to get socket descriptor for using ioctl() ");
      exit (EXIT_FAILURE);
   }

   // Use ioctl() to look up interface name and get its MAC address.
   memset (&ifr, 0, sizeof (ifr));
   snprintf (ifr.ifr_name, sizeof (ifr.ifr_name), "%s", interface);
   if (ioctl (sd, SIOCGIFHWADDR, &ifr) < 0) {
      perror ("ioctl() failed to get source MAC address ");
      return (EXIT_FAILURE);
   }
   close (sd);

   // Copy source MAC address.
   memcpy (src_mac, ifr.ifr_hwaddr.sa_data, 6);

   // Report source MAC address to stdout.
   printf ("MAC address for interface %s is ", interface);
   for (i=0; i<5; i++) {
      printf ("%02x:", src_mac[i]);
   }
   printf ("%02x\n", src_mac[5]);

   // Find interface index from interface name and store index in
   // struct sockaddr_ll device, which will be used as an argument of sendto().
   memset (&device, 0, sizeof (device));
   if ((device.sll_ifindex = if_nametoindex (interface)) == 0) {
```

```
    perror ("if_nametoindex() failed to obtain interface index ");
    exit (EXIT_FAILURE);
  }
  printf ("Index for interface %s is %i\n", interface, device.sll_ifindex);

  // Set destination MAC address: you need to fill these out
  dst_mac[0] = 0x10;//设置目的网卡地址
  dst_mac[1] = 0x78;
  dst_mac[2] = 0xd2;
  dst_mac[3] = 0xc6;
  dst_mac[4] = 0x2f;
  dst_mac[5] = 0x89;

  // Fill out sockaddr_ll.
  device.sll_family = AF_PACKET;
  memcpy (device.sll_addr, src_mac, 6);
  device.sll_halen = htons (6);

  // 发送的 data，长度可以任意，但是抓包时看到最小数据长度为 46，这是以太网协议规
定以太网帧数据域部分最小为 46 字节，不足的自动补零处理
  datalen = 12;
  data[0] = 'h';
  data[1] = 'e';
  data[2] = 'l';
  data[3] = 'l';
  data[4] = 'o';
  data[5] = ' ';
  data[6] = 'w';
  data[7] = 'o';
  data[8] = 'r';
  data[9] = 'l';
  data[10] = 'd';
  data[11] = '!';

  // Fill out ethernet frame header.
  frame_length = 6 + 6 + 2   + datalen;
  // Destination and Source MAC addresses
  memcpy (ether_frame, dst_mac, 6);
  memcpy (ether_frame + 6, src_mac, 6);

  ether_frame[12] = ETH_P_DEAN / 256;
  ether_frame[13] = ETH_P_DEAN % 256;

  // data
```

```
    memcpy (ether_frame + 14 , data, datalen);

    // Submit request for a raw socket descriptor.
    if ((sd = socket (PF_PACKET, SOCK_RAW, htons (ETH_P_ALL))) < 0) {//创建正真发送
的 socket
        perror ("socket() failed ");
        exit (EXIT_FAILURE);
    }
    // Send ethernet frame to socket.
    if ((bytes = sendto (sd, ether_frame, frame_length, 0, (struct sockaddr *) &device,
sizeof (device))) <= 0) {
        perror ("sendto() failed");
        exit (EXIT_FAILURE);
    }
    printf ("send num=%d,read num=%d\n",frame_length,bytes);
    // Close socket descriptor.
    close (sd);


    return (EXIT_SUCCESS);
}
```