HelloX网络功能简介及使用和开发指南

## HelloX网络功能简介

作为物联网操作系统，网络功能是必备的核心功能之一。按照规划，HelloX实现了两个不同类型的TCP/IP协议栈，一个面向资源受限的嵌入式应用，移植了业界成熟使用的lwIP协议栈。该协议栈简洁明了，功能相对简单，同时专门面向嵌入式领域进行设计和优化，对硬件资源要求很低。另外一个协议栈来自BSD操作系统的协议栈，面向复杂的网络功能丰富的应用场景，比如家庭网关，物联网网关等。为了适应HelloX本身的机制，对BSD协议栈做了一些更改和优化，当然，这是在BSD License允许的范围之内。

目前lwIP协议栈已经成熟稳定运行，面向复杂场景的BSD协议栈正在移植当中。不论采用哪个协议栈，都提供相同的，符合BSD规范的socket API，因此对应用程序来说，协议栈都是透明的。对网卡驱动来说，HelloX本身设计了一套网卡驱动机制，向上屏蔽了不同的IP协议栈之间的差异，因此不论安装的是lwIP协议栈还是BSD协议栈，网卡驱动也都是相同的。

本文对HelloX的网络功能的使用和开发方法，进行简单描述。

## 在虚拟机上使用网络功能

HelloX V1.78实现了一个PCNet Fast Ethernet III的网卡驱动程序（对应的网卡控制芯片是AMD的AM79C973）。大部分虚拟机都支持这个网卡，因此有比较广泛的通用性。如果您想试验其它的网卡驱动程序，则需要手工编写一个网卡驱动了。

以Virtual Box虚拟机为例，在虚拟机的“设置”界面中，选择“网络”，这时候会列举出四个网卡。选择“网卡1”，按下图所示进行配置：



注意，“连接方式”选择“桥接网卡”。如果选择其它连接类型，我没有试过，不知道行不行。选择桥接网卡的含义，就是把你的PC机的物理网卡，与虚拟机的“网卡1”，用二层的方式连接了起来。可以想象VirtualBox虚拟了一个以太网交换机，然后把你的电脑的物理网卡，以及虚拟机的网卡1，都连接在了这个虚拟的以太网交换机上。

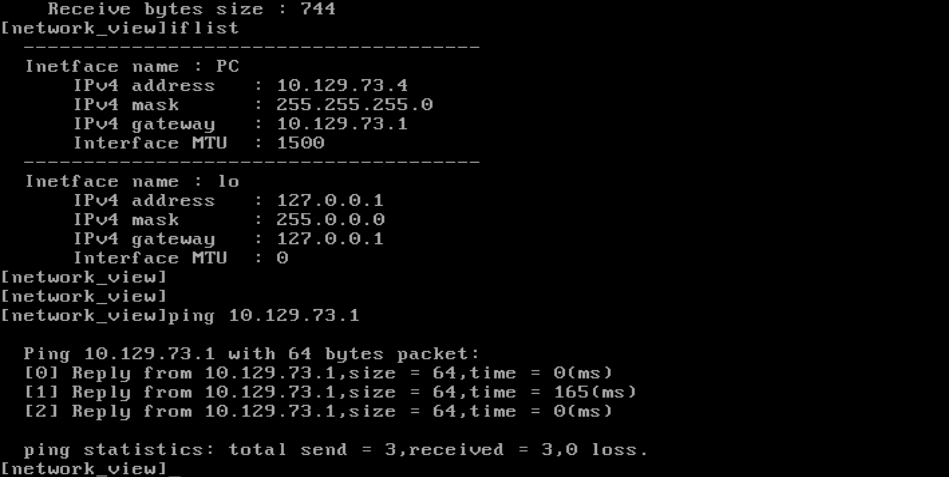
“界面名称”中，选择你的计算机的物理网卡名称。注意，这个网卡一定是可以用的网卡，最好是你当前上网用的网卡，如果有多个网卡的话。

点开“高级”选项，“控制芯片”一行，按照上图所示进行选择。如果选择其它的控制芯片，就需要编写对应控制芯片的驱动程序了。这是一项很有意思的工作，只要拿到对应芯片的data sheet，基本很快就可以编写出来。如果你能够成功的编写一个网卡驱动，那么恭喜你，应聘任何一家通信类或者嵌入式开发类公司，保准可以通过。有兴趣的朋友可以试试。

点击确定之后，用最新的HelloX版本启动虚拟机即可。注意，缺省情况下，HelloX是通过DHCP方式获取IP地址的，因此与HelloX虚拟机关联的物理网卡，也必须是通过DHCP方式获取IP地址。

当然，如果不是DHCP获取IP地址，也可以手工配置，参考本文下面的network命令部分。

启动虚拟机之后，就可以用network程序，来查看和诊断网络了。比如，可以用iflist命令，来查看获取到的IP地址，用ping命令，来试试网络连接是否正常。下面是在我的计算机上的运行结果：



在这里，简单介绍一下用ping命令诊断网络连接是否正常的方法。这些方法，一般在《网络基础》课程里面都会涉及到。如果要考MCSE，CCNA等证书，这些内容是必考内容。

首先，ping一下环回地址，即127.0.0.1。如果这个地址能ping通，说明TCP/IP协议栈已经正常工作了。如果ping不通，那么就是TCP/IP协议栈都没有起来；

如果环回地址ping通了，再ping一下网络接口的本地地址，也就是网卡地址。在上图中，是10.129.73.4。如果这个地址能够ping通，说明网卡已经被TCP/IP协议栈成功识别和管理。如果这个地址都ping不通，那么说明网卡与IP协议栈没有关联起来；

在本地地址ping通的情况下，再ping缺省网关，在上图中，就是10.129.73.1这个地址。如果这个地址能ping通，说明计算机与路由器之间的通信是正常的。后面如果有网络问题，基本上可以排除是本地计算机问题。如果ping不通，则说明本机与路由器没有正常通信，这时候需要检查网线等物理连接。

在缺省网关ping通的情况下，就可以ping目标服务器了。这时候如果ping通了，就万事大吉了。如果ping不通，那么很可能就是目标服务器的问题，或者是目标服务器与本地路由器之间的问题了。

## HelloX的Network程序使用指南

HelloX实现了一个network字符界面应用程序，可以对网络功能进行调试。同时实现了一个抽象的以太网管理框架（Ethernet Framework），实现了一套标准的网络驱动程序接口，屏蔽了不同网络驱动程序之间的差异。这样，不同的硬件，其驱动代码是不同的，但是只要遵循这一套标准的接口规范，就可以无缝挂接到HelloX内核中。

下面简要介绍V1.78版的网络调试程序network，在此基础上，简要介绍一下HelloX的网络驱动程序编写方法。

在字符shell模式下，输入network并回车，即可进入network应用程序。该程序提供了如下一些网络相关命令：

**scan命令**

该命令用于扫描所有可用的WiFi热点，前提是需要有WLAN硬件支持。该命令会列出所有扫描到的AP热点，如下：

[network-view]scan

Available WiFi list:

-----------------------------

00:BSSID = 20004E9C, RSSI = 76, SSID = 'HelloX\_HGW\_AP', channel = 1

01:BSSID = 2000508C, RSSI = 15, SSID = 'Celleden\_Map1600', channel = 6

如果发现想要连接的热点不在上述列表中，可多执行几次scan命令，很多情况下，一次scan是无法扫描到所有热点的。

注意，要使用scan命令，必须具备WLAN网卡，否则无效。

**assoc命令**

assoc用于跟某个指定的WiFi热点相关联。Scan只是扫描出一些可用的热点，但是如果需要跟某个热点进行连接，则必须使用assoc命令。如下：

[network-view]assoc HelloX\_HGW\_AP

上述命令用于连接到名字为“HelloX\_HGW\_AP”的WiFi热点。注意，该WiFi热点必须是开放的不加密热点，因为上述命令没有指定连接密码。

如果是要连接一个加密的WiFi热点，则可以用/k参数指定连接的密码：

[network-view]assoc HelloX\_HGW\_AP /k0123456789012

后面是密码。当前只支持WEP加密，因此密码必须是13个字节。

再次说明一下，当前值支持开放不加密的WiFi热点，以及基于WEP加密的WiFi热点，尚不支持WPA加密热点，因此要希望连接成功，必须修改WiFi热点的配置，修改为开放不加密，或者使用WEP加密（密码要设置为13位数字）。

**showint命令**

该命令用于显示出系统中所有网络接口的统计信息，比如接收报文个数，发送报文个数等。下列是一个简单的输出例子：

[network\_view]showint

Statistics information for interface 'Marvel\_WLAN\_Int':

Send frame # : 17

Success send # : 17

Send bytes size : 2946

Receive frame # : 14

Success recv # : 14

Receive bytes size : 1778

各输出字段的含义是自解释的。

**iflist命令**

能够则列举出系统中所有的网络接口，与其对应的IP地址等信息。与showint不同的是，iflist显示的是网络接口的静态信息（IP地址/掩码/缺省网关等），而showint显示的则是网络接口的动态信息。

下面是iflist的一个输出例子：

[network\_view]iflist

--------------------------------------

Inetface name : Ma

IPv4 address : 192.168.43.173

IPv4 mask : 255.255.255.0

IPv4 gateway : 192.168.43.1

Interface MTU : 1500

--------------------------------------

Inetface name : lo

IPv4 address : 127.0.0.1

IPv4 mask : 255.0.0.0

IPv4 gateway : 127.0.0.1

Interface MTU : 0

上面显示了两个网络接口，第一个是WLAN接口，名字只包含了起始的2个字节。

**setif命令**

setif命令则用于修改接口的静态配置参数。比如，可以通过下列方式，为一个网络接口设置静态的IP地址：

setif Marvel\_WLAN\_Int /a 192.168.0.100 /m 255.255.255.0 /g 192.168.0.1

其中Marvel\_WLAN\_Int是接口的名字（showint命令可以显示），后面分别是该接口的IP地址/子网掩码/缺省网关。需要注意的是，缺省情况下，接口上是启动DHCP功能的，试图自动获取IP地址。一旦通过上述命令设置静态IP地址，则会同时把该接口上的DHCP功能关闭。

如果要重新打开DHCP功能，则使用如下命令：

setif Marvel\_WLAN\_Int /d enable

而下列命令，则用于重新启动接口上的DHCP功能：

setif Marvel\_WLAN\_Int /d enable

重启DHCP功能的目的，是为了立即在接口上发出DHCP请求。缺省情况下，DHCP功能是以指数退避方式来发送DHCP请求报文的，即当接口刚刚使能的时候，会发送DHCP请求，如果没有收到响应，则会在2秒后再发一次，然后是4秒，然后是8秒…以此类推。如果想立即在接口上重新发出DHCP请求，则使用上述命令restart一下即可。

**Ping命令**

这是最常用的诊断命令，后面直接跟IP地址即可。下面是一个简单的输出例子：

[network\_view]ping 192.168.43.1

Ping 192.168.43.1 with 64 bytes packet:

[0]Reply from 192.168.43.1,size = 64,time = 40(ms)

[1]Reply from 192.168.43.1,size = 64,time = 180(ms)

[2]Reply from 192.168.43.1,size = 64,time = 180(ms)

ping statistics: total send = 3,received = 3,0 loss.

如果希望改变缺省的ping报文长度，则可以增加一个参数：

ping 192.168.43.1 /l 1024

上述命令以1024字节为ping报文长度。缺省情况下，会连续ping三个报文，然后结束。如果希望ping更多的报文，则使用下列命令：

ping 192.168.43.1 /c 1000

上述命令可以ping1000个报文。当然，l参数和c参数可以一起使用。

## HelloX网络驱动程序编写方法

HelloX实现了一个基于线程轮询机制的以太网驱动程序框架，系统中有一个叫做eth\_thread的线程，定时（每隔100ms）轮询网卡驱动程序，试图接收数据帧。如果有合适的数据帧到达，则eth\_thread会把这个数据帧递交到IP层处理。

因此，要实现一个以太网驱动程序，需要遵循HelloX的以太网驱动框架，具体来说，就是要实现下列函数：

**初始化函数**

原型如下，这个函数在以太网驱动程序被加载的时候，会被HelloX调用，用于完成硬件的初始化功能。当然，如果不需要初始化，完全可以写成下列形式：

static BOOL Int\_Init(\_\_ETHERNET\_INTERFACE\*pInt)

{

returnTRUE;

}

硬件的初始化，还可以放在驱动程序的入口函数中，下面会提及。

**报文发送函数**

原型如下：

static BOOL SendFrame(\_\_ETHERNET\_INTERFACE\*pInt);

在IP层试图发送报文的时候，以太网驱动框架会调用这个函数。所发送的数据帧已经在IP层面准备好（包括源MAC地址/目的MAC地址等），存放在pInt对象的一个缓冲区中（如下代码），驱动程序只需要发送即可。

typedef struct tag\_\_ETHERNET\_INTERFACE{

char ethName[MAX\_ETH\_NAME\_LEN +1];

char ethMac[ETH\_MAC\_LEN];

char SendBuff[ETH\_DEFAULT\_MTU];//Sending buffer.

int buffSize;

\_\_ETH\_INTERFACE\_STATE ifState;

LPVOID pL3Interface;

LPVOID pIntExtension; //Privateinformation.

……

}\_\_ETHERNET\_INTERFACE;

其中SendBuff是存放待发送数据帧的缓冲区，buffSize是数据帧的长度，必须小于ETH\_DEFAULT\_MTU（1500）。

在SendFrame函数中，只需要操作硬件，把SendBuff中的内容送到物理网络上，然后返回即可。

**数据帧接收函数**

该函数原型如下，会被HelloX的以太网管理框架周期性的调用，以判断是否有数据帧到达：

static struct pbuf\*RecvFrame(\_\_ETHERNET\_INTERFACE\* pInt);

在这个函数中，硬件判断有数据帧到达，则需要创建一个pbuf，把数据帧从硬件缓冲区中拷贝到pbuf里面，然后返回这个pbuf。比如下面的实例代码：

static struct pbuf\*Marvel\_RecvFrame(\_\_ETHERNET\_INTERFACE\* pInt)

{

struct eth\_packet \*rx\_pkt =&pgmarvel\_priv->rx\_pkt;

struct pbuf \*p, \*q;

u16 len =0;

int l = 0;

char \*buffer = NULL;

p= NULL;

/\*Obtain the size of the packet and put it into the "len" variable. \*/

len= lbs\_rev\_pkt();

if(len > 0){

buffer= rx\_pkt->data;

/\*We allocate a pbuf chain of pbufs from the pool. \*/

p= pbuf\_alloc(PBUF\_RAW, len, PBUF\_POOL);

if(p != NULL){

for(q = p; q != NULL; q = q->next){

memcpy((u8\_t\*)q->payload,(u8\_t\*)&buffer[l], q->len);

l= l + q->len;

}

}

else

{

}

}

return p;

}

如果硬件判断没有数据帧到达，则只需要返回NULL即可。

**特定功能的控制函数**

原型如下：

static BOOL Eth\_Ctrl(\_\_ETHERNET\_INTERFACE\*pInt,DWORD dwOperation,LPVOID pData);

对于一些以太网的特定控制功能，比如设置MTU大小，修改速率，WiFi的扫描AP/附着AP等，通过这个函数实现。dwOperation指明了需要的操作。

如果没有特殊需要，也可以不需要实现该函数。建议的实现方式是，实现一个只返回TRUE的空函数，比如：

static BOOLMarvel\_Ctrl(\_\_ETHERNET\_INTERFACE\* pInt,DWORD dwOperation,LPVOID pData)

{

returnTRUE;

}

**实现驱动程序入口函数**

实现完成上述函数之后，还需要实现一个以太网驱动程序的入口函数，这个入口函数会被HelloX的以太网管理框架调用，用于加载以太网驱动程序。在入口函数中，需要做系列工作：

1． 初始化硬件；

2． 调用AddEthernetInterface，向系统中注册以太网接口。

下面是一个实现实例：

BOOL Marvel\_Initialize(LPVOID pData)

{

\_\_ETHERNET\_INTERFACE\*pMarvelInt = NULL;

char mac[ETH\_MAC\_LEN];

//初始化硬件，获得硬件的MAC地址，存放在mac数组中。

//调用AddEthernetInterface，注册接口。

pMarvelInt= EthernetManager.AddEthernetInterface(

MARVEL\_ETH\_NAME, //以太网的名字，任意字符串，不能包含空格。

&mac[0], //MAC地址。

NULL, //初始化函数的参数。

Int\_Init, //接口初始化函数，对应上述Int\_Init函数

SendFrame, //数据帧发送函数

RecvFrame, //数据帧接收函数

Eth\_Ctrl); //控制函数。

if(NULL== pMarvelInt)

{

returnFALSE;

}

returnTRUE;

}

调用AddEthernetInterface的时候，需要使用上面实现的四个操作函数作为参数。调用成功后，会返回一个以太网对象指针，可以保存起来，供后期卸载以太网接口的时候用。

**在以太网驱动入口数组中增加一项**

最后一步，就是在以太网驱动程序入口函数数组中增加一项，告诉操作系统以太网驱动程序的存在。这样操作系统在初始化的时候，就会调用驱动程序入口函数，加载驱动程序。入口函数数组位于network/ethernet/ethentry.c文件中，下面是一个示例：

\_\_ETHERNET\_DRIVER\_ENTRYEthernetDriverEntry[] =

{

#ifdef \_\_CFG\_NET\_MARVELLAN

{Marvel\_Initialize,NULL},

#endif

//Please add your ethernet driver's entry here.

{NULL,NULL}

};

其中Marvel\_Initialize是驱动程序入口函数，NULL是入口函数的参数，可以是任意指针。注意，该数组中的第一个网络接口，会被系统自动设置为缺省网络接口，即缺省网关所在的接口，或者HelloX默认路由所在的接口。

这样以太网驱动程序就编写完成了。建议把以太网驱动程序的代码，存放在driver/STM32目录下。重新编译HelloX，即可实现以太网驱动程序的加载。

以太网驱动程序加载成功后，即可食用network程序，进行诊断和调试。如果命令级别的诊断无法发现问题，则可以启用代码级的调试。具体来说，在ethif.h文件中，打开以太网调试开关（\_\_ETH\_DEBUG），重新编译并加载运行，即可输出网络运行过程中的更加详细的信息。

HelloX V1.78的代码，可以从github上下载：

github.com/hellox-project/HelloX\_OS

有任何问题，欢迎加入QQ群讨论：38467832