# README for LAB #1

Yuyang Zhou
2000013061

## 1 Implemented

The features implemented are `PT1, PT2`. and this document include the answer of `WT1`

## 2 WT1

1. there are 827 frame in total
2. ff:ff:ff:ff:ff:ff

The destination address is special because it is a boardcast message. The computer with MAC address 6a:15:0a:ba:9b:7c doesn't know which computer have IP address 10.0.0.88/10.0.0.24, and it boardcast the message to all computers, trying to find the one with IP address 10.0.0.88/10.0.0.24.

3. 0x15 in hexademical, 21 in demical

## 3 Code Arrangement

### macro.h

define the macros used in the following programs.

### constant.h

define the constants used in the following programs.

### type.h

define the specific types used in the following programs.

### name2mac.h

find the Mac Address(actually hardware address) of specific device

- `findMac(const char* device, uint8_t* mac_addr)`

  * find the Mac Address of specific device

  * @param device the name of the device request for Mac address

  * @param mac_addr the pointer to the memory used to save the Mac address

  * @return 0 on success, 1 on failure.

## device.h

Library supporting network device management.

- `checkDevice(const char* device)`

  * Check whether the device name exists in the network.

  * @param device Name of network device to check.

  * @return True on success , False on error.

- `addDevice(const char* device)`

  * Add a device to the library for sending / receiving packets .

  * @param device Name of network device to send / receive packet on .

  * @return A non - negative _device - ID_ on success , -1 on error .

- `findDevice(const char* device)`

  * Find a device added by 'addDevice'.

  * @param device Name of the network device .

  * @return A non - negative _device - ID_ on success , -1 if no such device was found .

In order to manage this, the code two structures `DeviceNode` and `DeviceManager`, and implements the following methods

- `DeviceNode`

  – `char* device_names` The name of the device

  – `pcap_t* receive_handler` The handler used to receive messages

  – `pcap_t* send_handler` The handler used to send messages

  – `frameReceiveCallback callback` The default callback function

  – `uint8_t mac_addr[8]` Mac address of the device

  – `int index` The index of the device

  – `bool isEqualDevice(const char* device)`

    * Check if the name of the device is equal to 'device'

    * @param device The name of device to check out

    * @return True on same, False on different

  – `void setCallback(frameReceiveCallback __callback__)`

    * Set up the callback function of the device

    * @param ___callback___ the pointer of callback function to set up

  - `int setDevice(const char* device)`

    * Initalize the deviceNode with name device

    * @param device The name of device used to set up

    * @return 0 on success, -1 on failure.

  - `int handInPacket(struct pcap_pkthdr* pkt_header, const u_char* framebuf)`

    * Handle the infomation of the package founc by libpcap

    * @param pkt_header the header of the packet found

    * @param framebuf the information of the packet found

    * @return 0 on success, -1 on failure.

- `DeviceManager`

  - `DeviceNode** device_list` The pointer to the piece of memory, to save the pointer of `DeviceNodes`

  - `int device_count` The number of devices in the manager

  - `int device_bound` The maximum number of devices can be saved now

  - `DeviceNode* operator [](const int index)`

    * Return the pointer of the index-th DeviceNode

    * @param index The index of device to find

  - `int addDevice(const char* device)`

    * Add a device to the library for sending / receiving packets .

    * @param device Name of network device to send / receive packet on .

    * @return A non - negative _device - ID_ on success , -1 on error .

  - `findDevice(const char* device)`

    * Find a device added by 'addDevice'.

    * @param device Name of the network device .

    * @return A non - negative _device - ID_ on success , -1 if no such device was found .

  - `count()`

    * @return The number of devices in the manager.

## packetio.h

Library supporting sending / receiving Ethernet II frames.

- `int sendFrame (const void * buf, int len, int ethtype, const void *destmac, int id)`

  * Encapsulate some data into an Ethernet II frame and send it .

  * @param buf Pointer to the payload .

* @param len Length of the payload .

* @param ethtype EtherType field value of this frame .

* @param destmac MAC address of the destination .

* @param id ID of the device ( returned by ”addDevice”) to send on .

* @return 0 on success , -1 on error .

- `int setFrameReceiveCallback(frameReceiveCallback callback, int id)`

  * Register a callback function to be called each time an Ethernet II frame was received .

  *@param callback the callback function.

  * @return 0 on success , -1 on error.

- `int receiveAllFrame(int id, int frame_count)`

  * try to receive specific number of Ethernet frames from device ID id.

  * @param id The Index of device to receive the package.

  * @param frame_count A number,-1 represents receiving until error occurs, 0-65535 represents the number of packet expected to receive.

  * @return the number of packages received,

### main.cpp

- `msg` test message to send.

- `egCallback(const void* __buffer, const void* __mac_addr, int len, int index))`

  * Example Callback function.

  * @param ___buffer the message from the packet.

  * @param ___mac_addr the mac address of the source of the packet.

  * @param len the length of ___buffer

  * @param the index of device which receive the packet.

  * @return 0 on success, 1 on failure.

## 4 Implementing Details

### name2mac.h

The piece of code use `socket` and `ioctl` to find the MAC address, instead of directly reading files from `/etc/ether`.

And because of that, the code require root access to run, otherwise it will fail to find the MAC address.

### device.h

This piece of code use structure `DeviceNodes` to maintain all informations about specific devices, and this may helps to reduce the difficulty of the following labs. And also, each `DeviceNodes` use two different handlers, one for sending packets, and the other one for receiving packets. The receiver use a filter, so that it only react the packet which the destination address is the same as the mac address of the device. Also, there may have different callback function for different devices, so I choose to save the function for each device.

`DeviceManager` maintain a list, which save the pointer of all `DeviceNodes`. I use a hand-written memory management instead of directly `vector<DeviceNode>`, because the copy of `DeviceNode` may cause serious memory leaking. So I copy the pointer, instead of copy the structure itself.

In addition, `checkDevice` gives a interface, which can check that whether the specific device exists. And It will fail to add a device, if the manager could not find it.

### packetio.h

Because we have find the MAC address of the device, we can simply splicing all information we have to obtain the raw Ethernet frame. And the frame could be send through the `send-handler` we have create in `addDevice`.

In receiveAllFrame, Because we have set the filter `'ether dst xx:xx:xx:xx:xx:xx'`, we can find and only find the frame which is send to the specific device. The `frame_count` parameter, like `pcap_loop`, is designed for further labs. And because we have set non-blocking for the receiving handler, we must return the number of frame we have received due to delays.

### main.cpp

This part gives a example `egCallback()` function, which helps me to check whether the packets received are correctly handled.

The main function firstly print the name, and discription about all devices in the ethernet, and the MAC address of the devices if exists. Then it will insert all devices with MAC address into the device manager. After that it will try to send a raw Ethernet frame from device 0 to device 0, where device 0, the first device we insert, may be `ens33`(In my conputer) or `eth0`(Other computers). At last, it will catch 5 frames to device 0, and call `egCallBack()` 5 times, which print the information about the frame.

## 5 Usage

The executable file is generated using `cmake`.

```
mkdir build && cd build
cmake ..
make
```

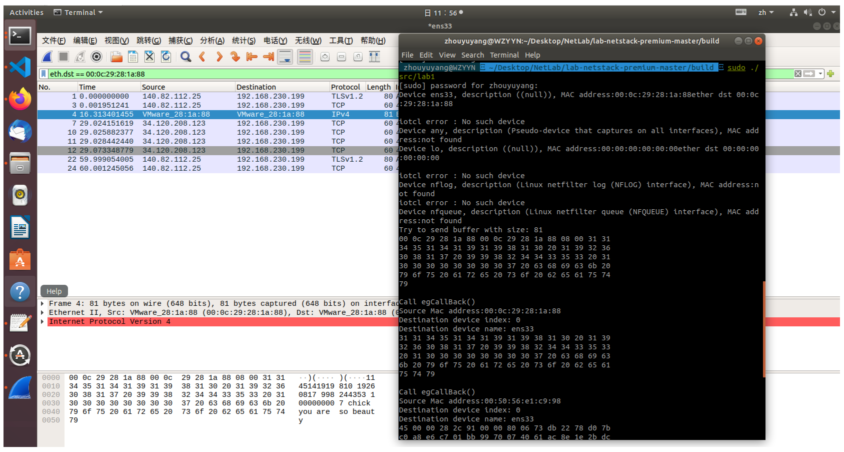
Then executable file is saved in `/build/src/lab1`, and you can use the following command to run it.

```
sudo ./src/lab1
```

## 6 CheckPoint

### CP1

The following image shows that we try to send a message `msg` which is defined in `main.cpp`. After senging out the message, the packet is catched by wireshark, and the frame if sent correctly.



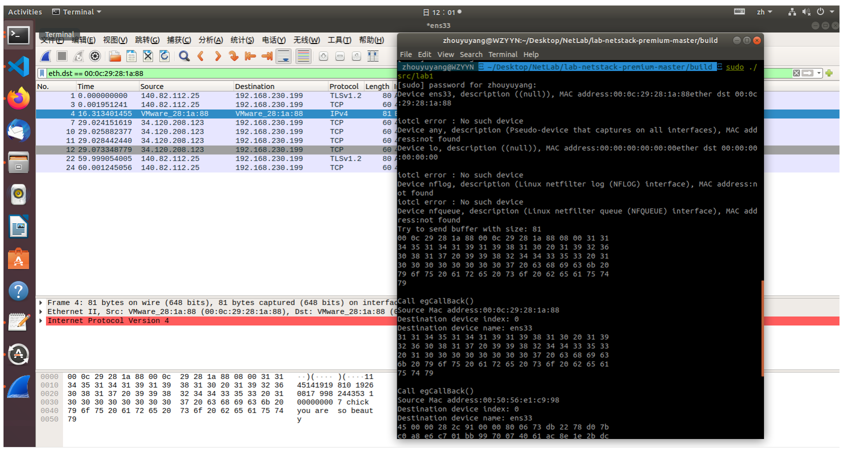
```
eth.dst == 00:0c:29:28:1a:88

No.  Time          Source          Destination      Protocol Length
   1 0.000000000   140.82.112.25   192.168.230.199  TLSv1.2      80
   3 0.001951241   140.82.112.25   192.168.230.199  TCP          60
   4 16.313401455  VMware_28:1a:88 VMware_28:1a:88  IPv4         81
   7 29.024151619  34.120.208.123  192.168.230.199  TCP          60
  10 29.025882377  34.120.208.123  192.168.230.199  TCP          60
  11 29.028442440  34.120.208.123  192.168.230.199  TCP          60
  12 29.073348779  34.120.208.123  192.168.230.199  TCP          60
  22 59.999054005  140.82.112.25   192.168.230.199  TLSv1.2      80
  24 60.001245056  140.82.112.25   192.168.230.199  TCP          60

▸ Frame 4: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interfac
▸ Ethernet II, Src: VMware_28:1a:88 (00:0c:29:28:1a:88), Dst: VMware_28:1a:88 (
▸ Internet Protocol Version 4

0000  00 0c 29 28 1a 88 00 0c  29 28 1a 88 08 00 31 31   ··)(···· )(····11
0010  34 35 31 34 31 39 31 39  38 31 30 20 31 39 32 36   45141919 810 1926
0020  30 38 31 37 20 39 39 38  32 34 34 33 35 33 20 31   0817 998 244353 1
0030  30 30 30 30 30 30 30 30  37 20 63 68 69 63 6b 20   00000000 7 chick
0040  79 6f 75 20 61 72 65 20  73 6f 20 62 65 61 75 74   you are  so beaut
0050  79                                                 y
```

```
zhouyuyang@WZYYN ~/Desktop/NetLab/lab-netstack-premium-master/build sudo ./
src/lab1
[sudo] password for zhouyuyang:
Device ens33, description ((null)), MAC address:00:0c:29:28:1a:88ether dst 00:0c
:29:28:1a:88

iotcl error : No such device
Device any, description (Pseudo-device that captures on all interfaces), MAC add
ress:not found
Device lo, description ((null)), MAC address:00:00:00:00:00:00ether dst 00:00:00
:00:00:00

iotcl error : No such device
Device nflog, description (Linux netfilter log (NFLOG) interface), MAC address:n
ot found
iotcl error : No such device
Device nfqueue, description (Linux netfilter queue (NFQUEUE) interface), MAC add
ress:not found
Try to send buffer with size: 81
00 0c 29 28 1a 88 00 0c 29 28 1a 88 08 00 31 31
34 35 31 34 31 39 31 39 38 31 30 20 31 39 32 36
30 38 31 37 20 39 39 38 32 34 34 33 35 33 20 31
30 30 30 30 30 30 30 30 37 20 63 68 69 63 6b 20
79 6f 75 20 61 72 65 20 73 6f 20 62 65 61 75 74
79

Call egCallBack()
Source Mac address:00:0c:29:28:1a:88
Destination device index: 0
Destination device name: ens33
31 31 34 35 31 34 31 39 31 39 38 31 30 20 31 39
32 36 30 38 31 37 20 39 39 38 32 34 34 33 35 33
20 31 30 30 30 30 30 30 30 30 37 20 63 68 69 63
6b 20 79 6f 75 20 61 72 65 20 73 6f 20 62 65 61
75 74 79

Call egCallBack()
Source Mac address:00:50:56:e1:c9:98
Destination device index: 0
Destination device name: ens33
45 00 00 28 2c 91 00 00 80 06 73 db 22 78 d0 7b
c0 a8 e6 c7 01 bb 99 70 07 40 61 ac 8e 1e 2b dc
```

Figure 1: CP-1: The program send a frame, and it is catched by wireshark

### CP2

The following image shows that the frame we send is successfully to `egCallBack()`

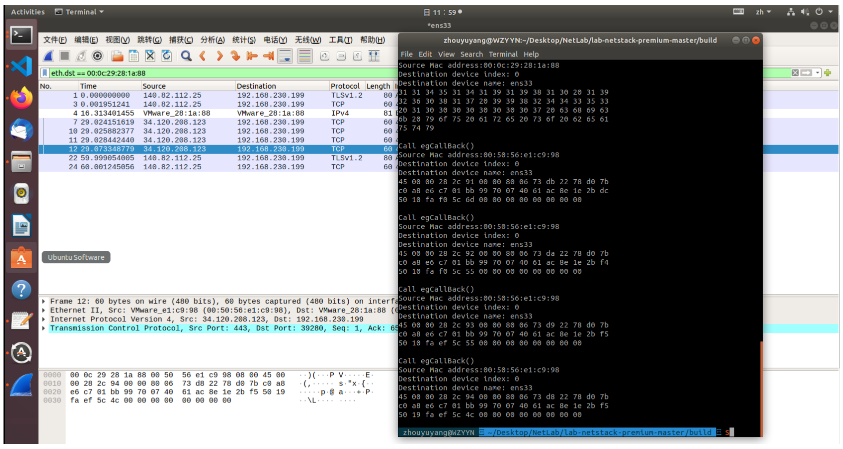


```
eth.dst == 00:0c:29:28:1a:88

No.  Time          Source          Destination      Protocol Length
   1 0.000000000   140.82.112.25   192.168.230.199  TLSv1.2      80
   3 0.001951241   140.82.112.25   192.168.230.199  TCP          60
   4 16.313401455  VMware_28:1a:88 VMware_28:1a:88  IPv4         81
   7 29.024151619  34.120.208.123  192.168.230.199  TCP          60
  10 29.025882377  34.120.208.123  192.168.230.199  TCP          60
  11 29.028442440  34.120.208.123  192.168.230.199  TCP          60
  12 29.073348779  34.120.208.123  192.168.230.199  TCP          60
  22 59.999054005  140.82.112.25   192.168.230.199  TLSv1.2      80
  24 60.001245056  140.82.112.25   192.168.230.199  TCP          60

▸ Frame 4: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interfac
▸ Ethernet II, Src: VMware_28:1a:88 (00:0c:29:28:1a:88), Dst: VMware_28:1a:88 (
▸ Internet Protocol Version 4

0000  00 0c 29 28 1a 88 00 0c  29 28 1a 88 08 00 31 31   ··)(···· )(····11
0010  34 35 31 34 31 39 31 39  38 31 30 20 31 39 32 36   45141919 810 1926
0020  30 38 31 37 20 39 39 38  32 34 34 33 35 33 20 31   0817 998 244353 1
0030  30 30 30 30 30 30 30 30  37 20 63 68 69 63 6b 20   00000000 7 chick
0040  79 6f 75 20 61 72 65 20  73 6f 20 62 65 61 75 74   you are  so beaut
0050  79                                                 y
```

```
zhouyuyang@WZYYN ~/Desktop/NetLab/lab-netstack-premium-master/build sudo ./
src/lab1
[sudo] password for zhouyuyang:
Device ens33, description ((null)), MAC address:00:0c:29:28:1a:88ether dst 00:0c
:29:28:1a:88

iotcl error : No such device
Device any, description (Pseudo-device that captures on all interfaces), MAC add
ress:not found
Device lo, description ((null)), MAC address:00:00:00:00:00:00ether dst 00:00:00
:00:00:00

iotcl error : No such device
Device nflog, description (Linux netfilter log (NFLOG) interface), MAC address:n
ot found
iotcl error : No such device
Device nfqueue, description (Linux netfilter queue (NFQUEUE) interface), MAC add
ress:not found
Try to send buffer with size: 81
00 0c 29 28 1a 88 00 0c 29 28 1a 88 08 00 31 31
34 35 31 34 31 39 31 39 38 31 30 20 31 39 32 36
30 38 31 37 20 39 39 38 32 34 34 33 35 33 20 31
30 30 30 30 30 30 30 30 37 20 63 68 69 63 6b 20
79 6f 75 20 61 72 65 20 73 6f 20 62 65 61 75 74
79

Call egCallBack()
Source Mac address:00:0c:29:28:1a:88
Destination device index: 0
Destination device name: ens33
31 31 34 35 31 34 31 39 31 39 38 31 30 20 31 39
32 36 30 38 31 37 20 39 39 38 32 34 34 33 35 33
20 31 30 30 30 30 30 30 30 30 37 20 63 68 69 63
6b 20 79 6f 75 20 61 72 65 20 73 6f 20 62 65 61
75 74 79

Call egCallBack()
Source Mac address:00:50:56:e1:c9:98
Destination device index: 0
Destination device name: ens33
45 00 00 28 2c 91 00 00 80 06 73 db 22 78 d0 7b
c0 a8 e6 c7 01 bb 99 70 07 40 61 ac 8e 1e 2b dc
```

Figure 2: CP2-1: The program receive the frame we send

The following image shows that the fifth frame we catch is exactly the fifth frame which is send to the device.


Activities
Terminal
11：59
*ens33
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
Ubuntu Software
File Edit View Search Terminal Help

```
eth.dst == 00:0c:29:28:1a:88

No.  Time          Source           Destination      Protocol  Length
   1 0.000000000   140.82.112.25    192.168.230.199  TLSv1.2   80
   3 0.001951241   140.82.112.25    192.168.230.199  TCP       60
   4 16.313401455  VMware_28:1a:88  VMware_28:1a:88  IPv4      81
   7 29.024151619  34.120.208.123   192.168.230.199  TCP       60
  10 29.025882377  34.120.208.123   192.168.230.199  TCP       60
  11 29.028442440  34.120.208.123   192.168.230.199  TCP       60
  12 29.073348779  34.120.208.123   192.168.230.199  TCP       60
  22 59.999054005  140.82.112.25    192.168.230.199  TLSv1.2   80
  24 60.001245056  140.82.112.25    192.168.230.199  TCP       60

▸ Frame 12: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interfa
▸ Ethernet II, Src: VMware_e1:c9:98 (00:50:56:e1:c9:98), Dst: VMware_28:1a:88 (0
▸ Internet Protocol Version 4, Src: 34.120.208.123, Dst: 192.168.230.199
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 39280, Seq: 1, Ack: 65

0000  00 0c 29 28 1a 88 00 50  56 e1 c9 98 08 00 45 00   ··)(···P V·····E·
0010  00 28 2c 94 00 00 80 06  73 d8 22 78 d0 7b c0 a8   ·(,····· s·"x·{··
0020  e6 c7 01 bb 99 70 07 40  61 ac 8e 1e 2b f5 50 19   ·····p·@ a···+·P·
0030  fa ef 5c 4c 00 00 00 00  00 00 00 00               ··\L···· ····
```

```
zhouyuyang@WZYYN:~/Desktop/NetLab/lab-netstack-premium-master/build
Source Mac address:00:0c:29:28:1a:88
Destination device index: 0
Destination device name: ens33
31 31 34 35 31 34 31 39 31 39 38 31 30 20 31 39
32 36 30 38 31 37 20 39 39 38 32 34 34 33 35 33
20 31 30 30 30 30 30 30 30 30 37 20 63 68 69 63
6b 20 79 6f 75 20 61 72 65 20 73 6f 20 62 65 61
75 74 79

Call egCallBack()
Source Mac address:00:50:56:e1:c9:98
Destination device index: 0
Destination device name: ens33
45 00 00 28 2c 91 00 00 80 06 73 db 22 78 d0 7b
c0 a8 e6 c7 01 bb 99 70 07 40 61 ac 8e 1e 2b dc
50 10 fa f0 5c 6d 00 00 00 00 00 00 00 00

Call egCallBack()
Source Mac address:00:50:56:e1:c9:98
Destination device index: 0
Destination device name: ens33
45 00 00 28 2c 92 00 00 80 06 73 da 22 78 d0 7b
c0 a8 e6 c7 01 bb 99 70 07 40 61 ac 8e 1e 2b f4
50 10 fa f0 5c 55 00 00 00 00 00 00 00 00

Call egCallBack()
Source Mac address:00:50:56:e1:c9:98
Destination device index: 0
Destination device name: ens33
45 00 00 28 2c 93 00 00 80 06 73 d9 22 78 d0 7b
c0 a8 e6 c7 01 bb 99 70 07 40 61 ac 8e 1e 2b f5
50 10 fa ef 5c 55 00 00 00 00 00 00 00 00

Call egCallBack()
Source Mac address:00:50:56:e1:c9:98
Destination device index: 0
Destination device name: ens33
45 00 00 28 2c 94 00 00 80 06 73 d8 22 78 d0 7b
c0 a8 e6 c7 01 bb 99 70 07 40 61 ac 8e 1e 2b f5
50 19 fa ef 5c 4c 00 00 00 00 00 00 00 00

zhouyuyang@WZYYN  ~/Desktop/NetLab/lab-netstack-premium-master/build  $
```

Figure 3: CP2-2: The fifth frame we receive

All these pictures can be found in `/checkpoints`.