Writing Task 1

1.

729

2.

The dst is broadcast (`ff.ff.ff.ff`). This frame is a DHCP discover frame.

3.

15

Programming Task 1

I implemented these function in `device.h, device.c`

```
/**
* 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.
*/
int addDevice(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.
*/
int findDevice(const char* device);

/**
 * Get the handle of a device by id.
 *
 * @param id the id to look up.
 * @return the pcap_t handle on success, NULL if fails.
*/
pcap_t *getDeviceHandle(int id);

/**
 * Get the name of a device by id.
 *
 * @param id the id to look up.
 * @return the name on success, NULL if fails.
*/
char *getDeviceName(int id);
```

```
/**
 * Detect local interfaces and print them.
*/
void showLocalInterfaces(void);
```

Programming Task 2

I implemented these function in `packetio.h, packetio.c`

```
/**
* @brief 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.
* @see addDevice
*/
int sendFrame(const void* buf , int len ,
uint16_t ethtype , const void* destmac , int id);

/**
 * A callback function to be used by pcap_loop() or pcap_next().
 *
 * @param deviceName the name of the device that receives a frame.
 * @param pkthdr the packet header of the frame.
 * @param packet the pointer to the packet.
*/
void printFrameInfo(unsigned char *deviceName, const struct pcap_pkthdr *pkthdr,
const unsigned char *packet);
```

Checkpoint1, Checkpoint2

```
make cp12; cd build;
sudo ./receiver
```

then `sudo ./sender` in another terminal.

The receiver first prints out the local interfaces(I mixed cp1 into cp2).

```
hotbuz@hotbuz:/mnt/d/storage/大学课程/ComputerNetworking/NetstackLab/lab-netstack-premium/build$ sudo ./receiver
pcap version = libpcap version 1.10.1 (with TPACKET_V3)
devices: 0, eth0, (null)
addr: 172.18.243.225, mask: 172.18.243.225
addr: 0.0.0.0, mask: 0.0.0.0
devices: 1, any, Pseudo-device that captures on all interfaces
devices: 2, lo, (null)
addr: 127.0.0.1, mask: 127.0.0.1
addr: 0.0.0.0, mask: 0.0.0.0
devices: 3, bluetooth-monitor, Bluetooth Linux Monitor
devices: 4, nflog, Linux netfilter log (NFLOG) interface
devices: 5, nfqueue, Linux netfilter queue (NFQUEUE) interface
devices: 6, dbus-system, D-Bus system bus
devices: 7, dbus-session, D-Bus session bus
```

Then the sender sends 5 broadcast ethernet II frames, and the receiver waits for 10 frames with filter "ether dst ff:ff:ff:ff:ff:ff", so we could see

```
eth0 receives a frame.
dest mac: ff:ff:ff:ff:ff:ff
source mac: 00:15:5d:51:59:1b

eth0 receives a frame.
dest mac: ff:ff:ff:ff:ff:ff
source mac: 00:15:5d:51:59:1b

eth0 receives a frame.
dest mac: ff:ff:ff:ff:ff:ff
source mac: 00:15:5d:51:59:1b

eth0 receives a frame.
dest mac: ff:ff:ff:ff:ff:ff
source mac: 00:15:5d:51:59:1b

eth0 receives a frame.
dest mac: ff:ff:ff:ff:ff:ff
source mac: 00:15:5d:51:59:1b
```

and if we run `sudo ./sender` once more the receiver turns off. It's likely that the receiver won't receive any other frames due to the filter "ether dst ff:ff:ff:ff:ff:ff".