Lab 2b报告

王颖 1900013016

Network-layer: IP Protocol

> 1. Update the method in device.h, Use the library in packetio.h to implement the following methods to support sending/receiving IP packets.

- See `incl/ip.h`,`src/ip.cpp` and `incl/routing.h`,`scr/routing.cpp`
- The following functions are implemented:

```
/**
* @file ip.h
* @brief Library supporting sending/receiving IP packets encapsulated in an
Ethernet II frame.
*/
#include <netinet/ip.h>
#include <arpa/inet.h>

#ifndef _IP_H_
#define _IP_H_

#define ETHTYPE_IPV4 0x0800
#define IPV4_MYARP 0xf0

struct ipHdr_t {
    uint32_t p1;
    uint32_t p2;
    uint8_t ttl;
    uint8_t protocol;
    uint16_t hdrCRC;
    uint32_t src;
    uint32_t dst;
};

uint32_t changeEndian32(uint32_t x);

//Merge c0, c1, c2, c3 together, so they can in-order in memory
//This mechina is little Endian one
uint32_t merge32(uint8_t* c);

/**
* @brief Send an IP packet to specified host.
*
* @param src Source IP address.
* @param dest Destination IP address.
* @param proto Value of `protocol` field in IP header.
* @param buf pointer to IP payload
* @param len Length of IP payload
* @return 0 on success, -1 on error.
*/
int sendIPPacket(const struct in_addr src, const struct in_addr dest, int proto,
const void *buf, int len, int ttl = 32);
```

```
/**
* @brief Process an IP packet upon receiving it.
*s
* @param buf Pointer to the packet.
* @param len Length of the packet.
* @return 0 on success, -1 on error.
* @see addDevice
*/
typedef int (*IPPacketReceiveCallback)(const void* buf, int len);

int recvPacket(const void* buf, int len);

/**
* @brief Register a callback function to be called each time an IP packet was
received.
*
* @param callback The callback function.
* @return 0 on success, -1 on error.
* @see IPPacketReceiveCallback
*/
int setIPPacketReceiveCallback(IPPacketReceiveCallback callback);

/**
* @brief forward a packet which is not sended to current host
*
* @param buf Pointer to the packet.
* @param len Length of the packet.
* @return 0 on success, -1 on error.
*/
int forward(const void* buf, int len);

#endif
```

```
#include <bits/stdc++.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <unistd.h>
using namespace std;

#ifndef _ROUTING_H_
#define _ROUTING_H_

struct rtValue {
    unsigned char dstMAC[6];
    int32_t deviceID;
    int32_t distance;
    int32_t timestamp;
    rtValue() {deviceID = -1; timestamp = 0; distance = 1e9; };
};

struct nxtHop {
    unsigned char dstMAC[6];
    int deviceID;
};
```

```
//rtKey: pair(dst, mask)
typedef pair <in_addr_t, in_addr_t> rtKey;

void initRoutingTable();

//find corresponding entry according to Longest Prefix Matching
vector <nxtHop> lookForRouting(const struct in_addr dest, const struct in_addr
mask);

//Broadcast your RoutingTable to neighbours
int sendDVR();

//Change your RoutingTable according to your neighbour's RoutingTable
int recvDVR(const void* buf, int len, int id, const void* MAC);

//Discard the expired entry of RoutingTable
int updateDVR();


void *routingThread(void *vargp);

#endif
```

> 2. sendFrame() requires the caller to provide the destination MAC address when sending IP packets, but users of IP layer will not provide the address to you. Explain how you addressed this problem when implementing IP protocol.

Using the RoutingTable, you can view the MAC address of nextHop on the path to the destination. The RoutingTable can be maintained in recvDVR() using srcMAC information provided by Link Layer.

> 3. Describe your routing algorithm.

- I implement Distance Vector Routing algorithm.
- RoutingTable maps (destIP, maskIP) entries to their counterparts (nextHopMAC, sent device ID, total hops number to destIP, timestamp).
- A thread is responsible for periodically cleaning up expired entries in the RoutingTable and broadcasting RoutingTable to neighbors. (see `routingThread()`)
- My algorithm tends to keep the entries updated by the information received in the latest time period, which is controlled by `src / routing.cpp/clocker`. For multiple equally new paths, the algorithm will choose the shorter one.
- For devices on the same host, special judgment will be performed when setting the routingtable table entry. It is considered that the distance is equal to 0.

> 4. Use tcpdump/wireshark to capture the IP packets generated by your implementation. Hexdump the content of any one packet here, and show meanings for each byte (for example, "the first byte is 0x50 and the most significant 4 bits of the first byte is 0101, it means ...; and ...").

```
22:15:39.059447 IP none44353-virtual-machine > 255.255.255.255:  ip-proto-240
196
    0x0000:  ffff ffff ffff c2de d312 10dc 0800 4400
    0x0010:  00d8 0000 0000 01f0 0000 0a64 0101 ffff
    0x0020:  ffff 0a64 0101 ffff ffff c2de d312 10dc
    0x0030:  0000 0000 0000 0000 0000 0200 0000 0a64
    0x0040:  0201 ffff ffff a2d4 6f7b 609d 0000 0000
```

```
0x0050:  0000 0100 0000 0200 0000 0a64 0301 ffff
0x0060:  ffff a2d4 6f7b 609d 0000 0000 0000 0200
0x0070:  0000 0200 0000 0a64 0102 ffff ffff a2d4
0x0080:  6f7b 609d 0000 0000 0000 0100 0000 0200
0x0090:  0000 0a64 0202 ffff ffff a2d4 6f7b 609d
0x00a0:  0000 0000 0000 0200 0000 0200 0000 0a64
0x00b0:  0302 ffff ffff a2d4 6f7b 609d 0000 0000
0x00c0:  0000 0300 0000 0200 0000 0a64 0101 ffff
0x00d0:  ffff a2d4 6f7b 609d 0000 ffff ffff 0000
0x00e0:  0000 0000 0000 0000 0000
```

The above is a package sent by my implementation and captured by tcpdump.

- The first 14 bytes correspond to the Ether header.
  - `ffff ffff ffff` is srcMAC
  - `c2de d312 10dc` is dstMAC
  - `0800` is Ethertype
- The next 16 bytes correspond to the IP header.
  - `4` is IP version number
  - `4` indicates that the header length is 4 * 4 bytes
  - `00` is Qos and ECN related
  - `00d8` indicates the total length of the IP packet
  - `0000 0000` is identifier, flags, fragmented offset used to fragment and reassemble packets.
  - `01` is TTL(time to live)
  - `f0` is protocol. I use `0xf0` to identify packets for maintaining RoutingTable
  - `0000` is header checksum. I don't implement it.
  - `0a64 0101` and `ffff ffff` is scrIP and dstIP
- Next, every 28 bytes represents a routingtable table item.
  - Take the first item as an example.

```
0a64 0101 ffff ffff c2de d312 10dc
0000 0000 0000 0000 0000 0200 0000
```

  - `0a64 0101` is destIP
  - `ffff ffff` is mask
  - `c2de d312 10dc` is MAC address of next hop
  - `0000` is for alignment
  - then `0000 0000` means the host can send frame on device 0
  - `0000 0000` means from current host to IP 10.100.1.1, the length of shortest path is 0.
  - `0200 0000` indicates that the time stamp of this entry is 2. (By default, the entry data is represented in the small end method)

> 5. Use vnetUtils or other tools to create a virtual network with the following topology and show that: ns1 --- ns2 --- ns3 --- ns4
>
>   5.1. ns1 can discover ns4;
>
>   5.2. after we disconnect ns2 from the network, ns1 cannot discover ns4;
>
>   5.3. after we connect ns2 to the network again, ns1 can discover ns4

I use `cp4.txt` to build the network.

Change the network structure according to the above requirements and observe the results of the routingtable on NS1 in turn. We have

```
updateDVR......
(0a:64:01:01,ff:ff:ff:ff:ff)---->(82:3f:75:7e:f3:60 dis=0 stamp=1 id=0
(0a:64:02:01,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=1 stamp=1 id=0
(0a:64:03:01,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=2 stamp=1 id=0
(0a:64:01:02,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=1 stamp=1 id=0
(0a:64:02:02,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=2 stamp=1 id=0
(0a:64:03:02,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=3 stamp=1 id=0
```

```
updateDVR......
----------------------------------------------------
```

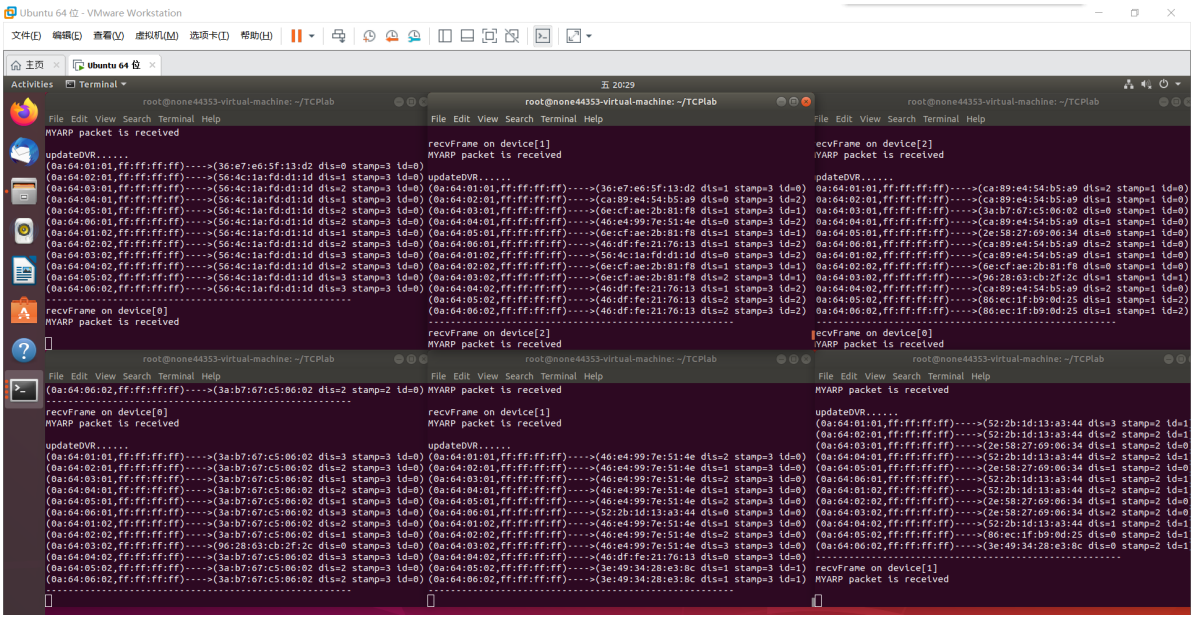
```
updateDVR......
(0a:64:01:01,ff:ff:ff:ff:ff)---->(82:3f:75:7e:f3:60 dis=0 stamp=1 id=0
(0a:64:02:01,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=1 stamp=1 id=0
(0a:64:03:01,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=2 stamp=1 id=0
(0a:64:01:02,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=1 stamp=1 id=0
(0a:64:02:02,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=2 stamp=1 id=0
(0a:64:03:02,ff:ff:ff:ff:ff)---->(d2:cb:4e:c4:24:7a dis=3 stamp=1 id=0
```

IP 0a:64:03:02 is device IP on NS4.

PS: In my implementation, the timestamp will rotate between 0-3, without sequential significance. (see `src/routing.cpp`)

> 6. Create a virtual network with the following topology and show the distances between each pair of hosts. The distance depends on your routing algorithm. After that, disconnect ns5 from the network and show the distances again.
>
> ns1 --- ns2 --- ns3 --- ns4
>
> | |
>
> ns5 --- ns6

I use `cp5.txt` to build the network. Observe the results of the routingtable on NS1 to NS6. We have

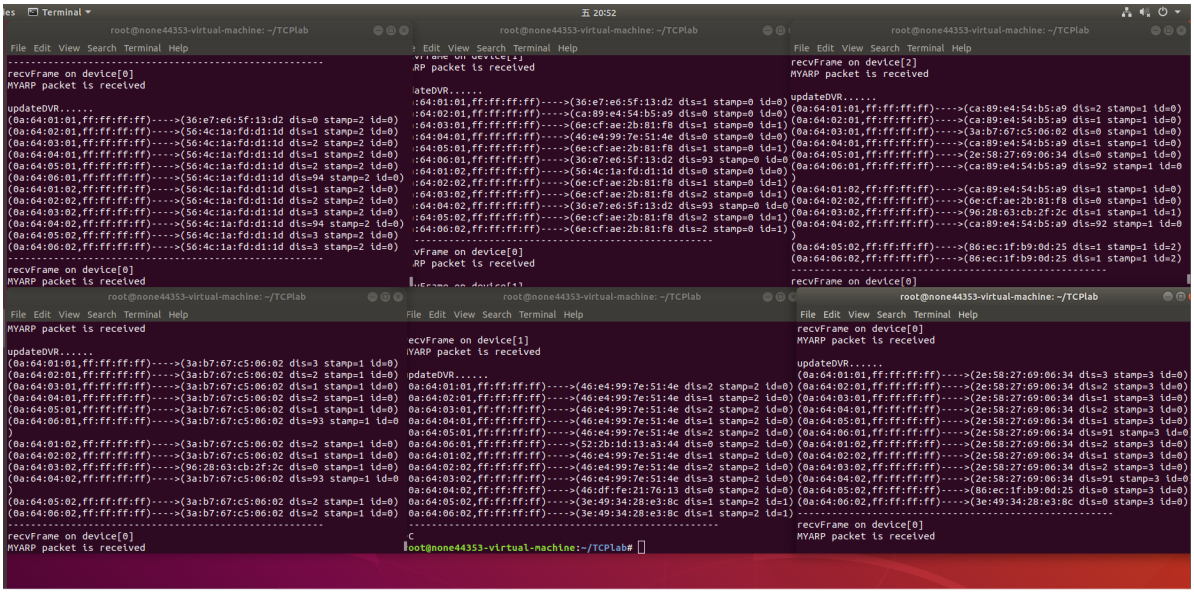


From top to bottom, from left to right are the terminals of NS1, NS2, NS3, NS4, NS5 and NS6.

- NS1 has IP address `0a:64:01:01`
- NS2 has IP address `0a:64:01:02`, `0a:64:02:01`, `0a:64:04:01`

- NS3 has IP address `0a:64:02:02`, `0a:64:03:01`, `0a:64:05:01`
- NS4 has IP address `0a:64:03:01`
- NS5 has IP address `0a:64:04:02`, `0a:64:06:01`
- NS6 has IP address `0a:64:05:02`, `0a:64:06:02`

After disconnect NS5, we have



The distance to IP of NS5 will get bigger and bigger over time.

> 7. Show the "longest prefix matching" rule applies in your implementation.

See the `lookForRouting()` function in `src/routing.cpp`.

I look up the qualified table item which has the longest mask in the routing table.

The following are the related implementations:

```
sem_wait(&rt_mutex);
map <rtKey, rtValue> :: iterator it = RoutingTable.begin();

in_addr_t preLst = 0;
for (it; it != RoutingTable.end(); ++it) {
    rtKey key = (*it).first;
    if ((key.first & key.second) != (dest.s_addr & key.second)
        || (key.second & mask.s_addr) != key.second) continue;

    preLst = max(preLst, key.second);
}

for (it; it != RoutingTable.end(); ++it) {
    rtKey key = (*it).first;
    if ((key.first & key.second) != (dest.s_addr & key.second)
        || (key.second & mask.s_addr) != key.second) continue;

    if (key.second != preLst) continue;

    rtValue value = (*it).second;
    nxtHop nhop;
    nhop.deviceID = value.deviceID;
    memcpy(nhop.dstMAC, value.dstMAC, 6);

    ret.push_back(nhop);
```

```
    preLst = 0x10; // In this case, only 1 nxthop is needed.
}

......

    sem_post(&rt_mutex);
return ret;
```

This complete part B.