中国科学院大学

University of Chinese Academy of Sciences

学生实验报告

课程名称：	计算机网络
培养单位：	声学研究所
学生学院：	电子电气与通信工程学院
修读专业：	信号与信息处理
学生学号：	202128001027007
学生姓名：	王鑫硕

日期：2021 年 11 月 9 日星期二

一、 实验题目

交换机转发实验

二、 实验目的

1. 学习交换机的工作原理；
2. 学习交换机转发和广播数据包的流程；
3. 学习转发表的数据结构；
4. 学习转发表的维护操作。

三、 实验环境

虚拟平台：Vmware Workstation Pro

操作系统：Ubuntu 16.04 LTS

gcc：5.4.0

python：2.7.12

Mininet：2.3.0

Wireshark：2.6.10

四、 实验内容

1. 在主机 h2 和 h3 上分别打开 wireshark 程序，监听各自主机的 eth0 端口（h2-eth0 和 h3-eth0）。
2. 在 h1 主机上分别 ping h2 和 h3 两个主机，在 h2 和 h3 两个主机上的 wireshark 捕获的应该只包含自己节点和 h1 产生的数据包。

五、 实验流程

1. 首先分析交换机中所包含的数据结构：

① ：端口信息结构体：iface_info_t

```c
typedef struct {
    struct list_head list;      // list node used to link all interfaces

    int fd;                     // file descriptor for receiving & sending
                                // packets
    int index;                  // the index (unique ID) of this interface
    u8  mac[ETH_ALEN];          // mac address of this interface
    char name[16];              // name of this interface
} iface_info_t;
```

该结构体存放了交换机上的端口信息，包括文件描述符、端口 ID、mac 地址、端口名称，并通过内核链表进行连接。

② ：mac 地址与端口映射表：mac_port_entry

```c
struct mac_port_entry {
    struct list_head list;
    uint8_t mac[ETH_ALEN];
    iface_info_t *iface;
    time_t visited;
};
```

该结构体存放了交换机记录的转发 mac 地址，与 mac 地址对应的转发端口信息，以及老化时间 visited，也是通过内核链表进行连接。

③ ：mac 地址与 hash 映射表：mac_port_map_t

```c
typedef struct {
    struct list_head hash_table[HASH_8BITS];
    pthread_mutex_t lock;
    pthread_t thread;
} mac_port_map_t;
```

该结构体通过内存放了 256 个 hash key，每个 hash key 通过内核链表串联起映射到自己的 mac 地址表（即上述第②个结构体），结构体内还包含互斥锁供多线程操作。

2. 实现对数据结构 mac_port_map 的所有操作，以及数据包的转发和广播操作。

①. 首先实现 iface_info_t *lookup_port(u8 mac[ETH_ALEN])函数，该函数的难点在于遍历 hash 表来找到对应的端口号，但内核链表给我们提供了宏函数 list_for_each_entry 来进行遍历操作。

```c
    pthread_mutex_lock(&mac_port_map.lock);     //Lock the map
    list_for_each_entry(entry, &mac_port_map.hash_table[index], list) {  //遍历对应hash表
        findFlag = 0;
        for(int i = 0; i < ETH_ALEN; i++){
            if(entry->mac[i] == mac[i])
                findFlag++;
            else
                break;
        }
        if(findFlag == ETH_ALEN){   // 找到了mac addr
            break;
        }
    }
    if(findFlag == ETH_ALEN){
        entry->visited = time(NULL);        // refresh time
        pthread_mutex_unlock(&mac_port_map.lock);   // unlock
        fprintf(stdout, "MyFlag: mac addr have benn found.\n");
        return entry->iface;
    }else{
        pthread_mutex_unlock(&mac_port_map.lock);   // unlock
        fprintf(stdout, "MyFlag: mac addr not found.\n");
        return NULL;
    }
}
```

我的实现过程是首先根据 mac 地址计算对应的 hash 值，然后通过 list_for_each_entry 宏函数对对应 hash 值下的链表进行遍历，遍历时加互斥锁，退出时解锁，通过 findFlag 作为找到对应 mac 地址的标志。若找到，则在退出时更新老化时间 visit,并返回转发端口;若未找到,则在退出时返回 null。

②. 接下来实现 void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)函数。

```c
void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)
{
    // TODO: implement the insertion process here
    fprintf(stdout, "TODO: implement the insertion process here.\n");
    mac_port_entry_t *insert_port;
    insert_port = (mac_port_entry_t*)malloc(sizeof(mac_port_entry_t));  //为端口信息开辟新空间
    for(int i = 0; i < ETH_ALEN; i++)       // 记录端口信息
        insert_port->mac[i] = mac[i];
    insert_port->iface = iface;
    insert_port->visited = time(NULL);

    u8 index = hash8(mac,ETH_ALEN);         // 计算hash值

    pthread_mutex_lock(&mac_port_map.lock);     // 上锁
    list_add_head(&insert_port->list, &mac_port_map.hash_table[index]); //  插入该端口
    pthread_mutex_unlock(&mac_port_map.lock);   // 解锁
}
```

该函数主要是对结构体 mac_port_entry_t 的操作，首先构建要插入的端口信息结构体，然后计算该收到包的 mac 地址对应的 hash 值，将该结构体插入到

对应 hash 值下的链表，该插入操作是通过内核链表提供的宏函数 list_add_head 所完成的，执行插入操作时，对映射表加互斥锁。

③. 接下来是 int sweep_aged_mac_port_entry()函数的实现。

```
int sweep_aged_mac_port_entry()
{
    // TODO: implement the sweeping process here
    pthread_mutex_lock(&mac_port_map.lock);
    int del_count = 0;
    mac_port_entry_t *entry, *q;
    for (int i = 0; i < HASH_8BITS; i++) {
        list_for_each_entry_safe(entry, q, &mac_port_map.hash_table[i], list) {
            if((int)(time(NULL) - entry->visited) > 30){
                del_count++;
                list_delete_entry(&entry->list);
                free(entry);  //非safe方法，此处free会异常
            }
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);
    return del_count;
}
```

该函数的难点在于遍历并删除超过 30 秒未访问的转发条目，通过内核链表提供的 list_for_each_entry_safe 宏函数对链表进行遍历，找到与当前时间差超过 30 的条目并执行删除操作，删除操作使用的也是系统提供的宏函数。在调试过程中发现，执行 free 操作必须要使用 safe 的遍历方法，否则会产生指针错误。

④. 接下来是 void broadcast_packet(iface_info_t *iface, char *packet, int len)函数的实现。

```
void broadcast_packet(iface_info_t *iface, char *packet, int len)
{
    // TODO: implement the broadcast process here
    iface_info_t *entry;
    list_for_each_entry(entry, &instance->iface_list,list){
        if(entry->fd != iface->fd){     //非发送端口
            iface_send_packet(entry,packet,len);    //广播该报文
        }
    }
    fprintf(stdout, "TODO: implement the broadcast process here.\n");
}
```

该函数要实现的是广播操作，通过 list_for_each_entry 函数进行遍历，找到非发送端口并将数据包发送出去，实现广播操作。

⑤. 最后是实现 void handle_packet(iface_info_t *iface, char *packet, int len)函数。

```c
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    // TODO: implement the packet forwarding process here
    struct ether_header *eh = (struct ether_header *)packet;
    log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_dhost));
    iface_info_t *dest = lookup_port(eh->ether_dhost);  //目标端口
    //先找目的地址
    if(dest != NULL){       //找到地址
        iface_send_packet(dest, packet, len);   //转发
    }else{                  //不存在该地址
        broadcast_packet(iface, packet, len);   //广播
    }
    //处理源端口地址
    iface_info_t *src = lookup_port(eh->ether_shost);   //源端口
    if(src == NULL) //未找到原地址
        insert_mac_port(eh->ether_shost,iface);
    fprintf(stdout, "TODO: implement the packet forwarding process here.\n");
}
```

该函数要实现的是包处理功能，收到数据包后首先获取包内的目的地址，通过 lookup_port 函数查找转发表中是否存在对应的转发端口，若存在，则通过对应端口转发，否则广播该数据包。之后获取数据包的源地址，在转发表中若未找到该地址的条目，则将其 mac 地址以及对应端口插入至转发表。

3. 基于 Ubuntu 系统，在 mininet 平台验证交换机程序的正确性

①. 首先通过 make 命令，对所有代码进行联合编译。

②. 打开终端，运行 sudo python2 three_nodes_bw.py 命令，启动老师给定的拓扑结构。

③. 在 mininet 平台上，通过 xterm s1 h2 h3 命令，启动 s1,h2,h3 的虚拟终端。

④. 在 s1 上运行./switch 命令，启动交换机程序。

⑤. 在 h2 和 h3 上运行 sudo wireshark 命令，启动 wireshark 软件，并监听各自的 eth0 端口的网络数据。至此的运行界面如图 1 所示。

⑥. 在 mininet 中运行 h1 ping h2 和 h1 ping h3 命令。

h1 ping h2 的结果如图 2 所示，在 h2-eth0 端口捕获到了源地址为 h1 目的地址为 h2 的数据包，同时，在 h3-eth0 端口没有捕捉到来自 h1 的数据包，只有 ARP 的广播地址包。

h1 ping h3 的结果如图 3 所示，在 h3-eth0 端口捕获到了源地址为 h1 目的地址为 h3 的数据包，同时，在 h2-eth0 端口没有捕捉到来自 h1 的数据包，只有 ARP 的广播地址包。

老化操作如图 4 所示，s1 主机展示了删除表项的信息（最后一行）。

实验结果表明，基本符合实验要求，实现了交换机对数据包的转发功能以及转发表的维护。

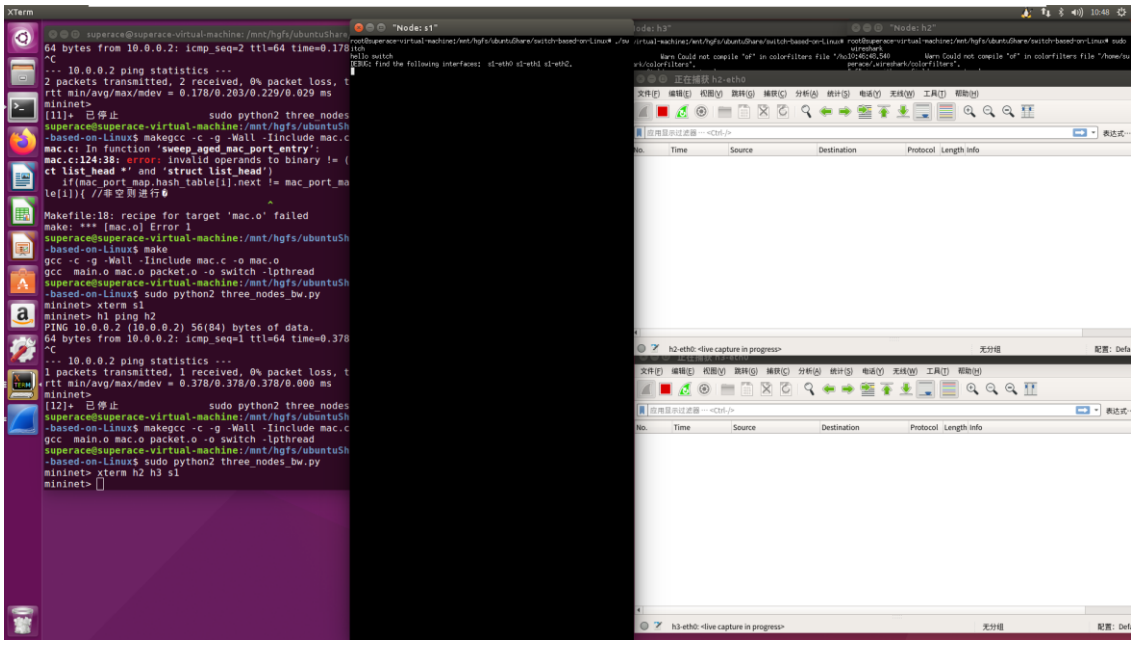

图 1 实验初始化环境展示

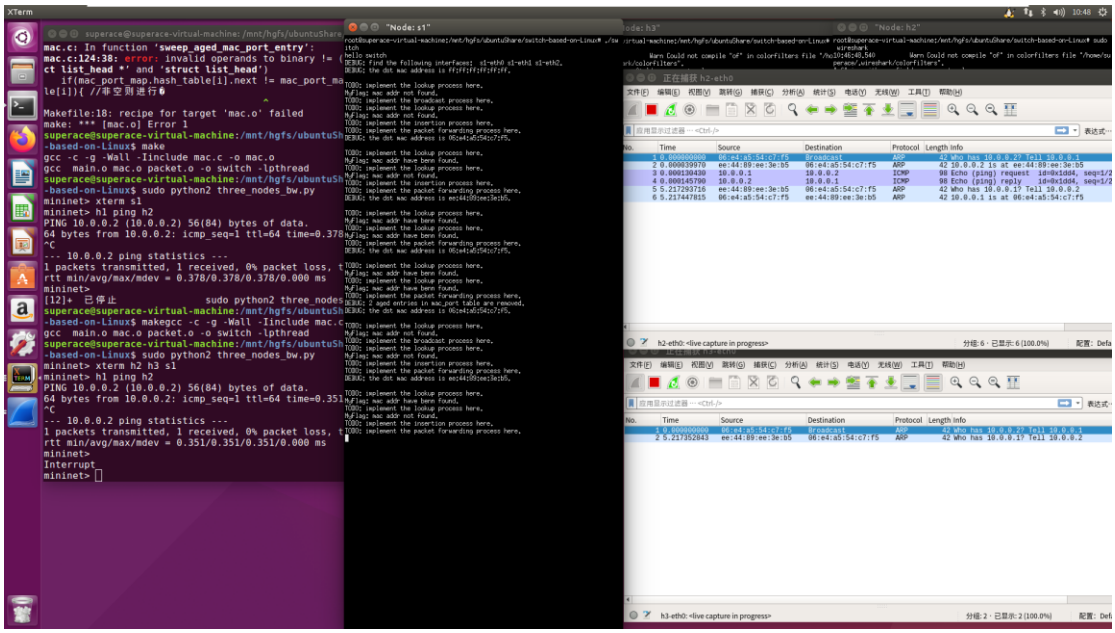

图 2 h1 ping h2 实验结果

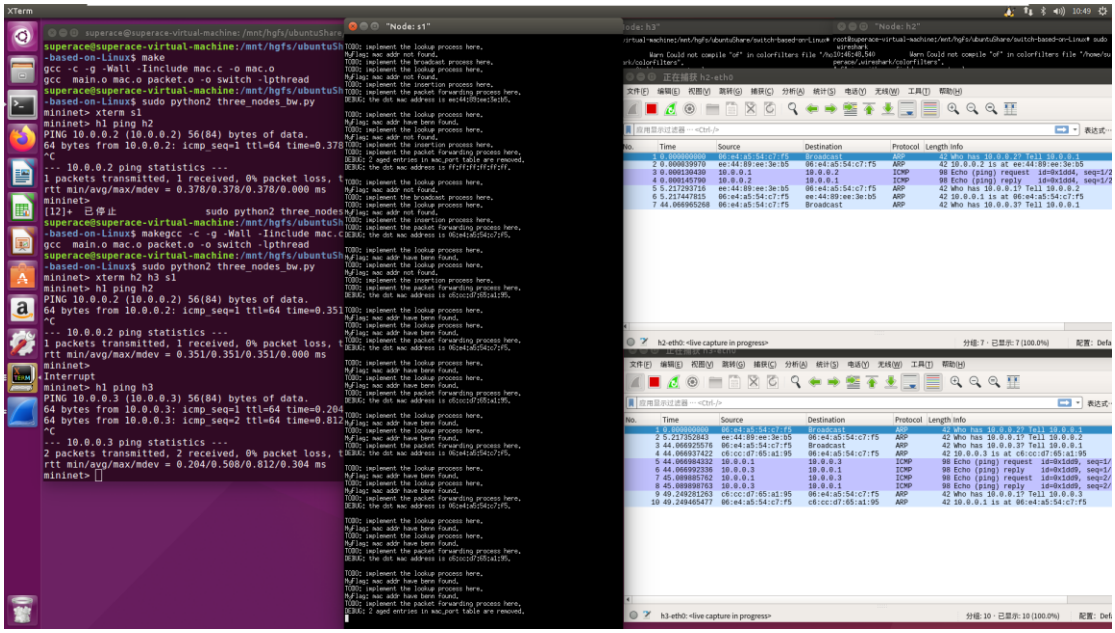

图 3 h1 ping h3 实验结果

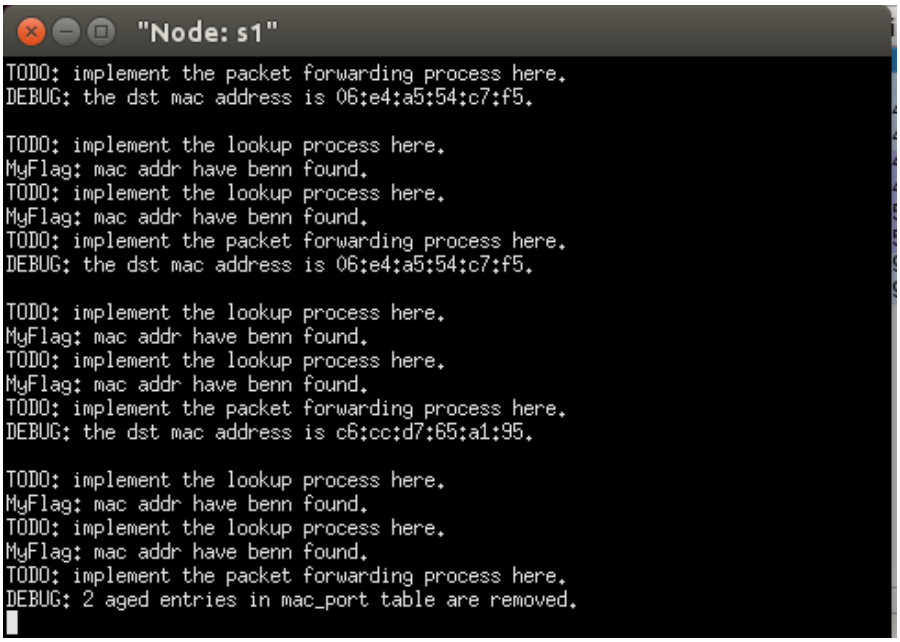

图 4 s1 交换机删除表项信息

六、 实验问题分析

1. 实验过程中遇到的第一个问题是内核链表的使用，通过资料的查找和学习，理解了其工作原理，通过计算结构体中链表的地址以及结构体的地址来找到结构体指针应该指向的首位置，以此来实现对结构体的访问以及修改等操作。

2. 代码编写过程中，在遍历链表时卡了一段时间，主要原因是因为先使用了系统提供的宏函数 list_for_each_entry(),并在遍历过程中 free()了传入的指针，之后查找了更多的资料后，发现了系统也提供了一种 safe 的遍历方法，即多借用一个中转指针来保存遍历指针位置，可以安全的调用 free()方法且不会引起错误。

七、 心得体会

通过此次实验，我系统地学习了交换机的工作原理，包括交换机收到一个数据包的工作流程、交换机中转发表的数据结构、转发表的维护等内容。帮助我更好地理解了交换机在计算机网络系统中的位置，以及所能提供的服务细节。