

**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

1. **实验题目**

交换机转发实验

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

虚拟平台：Vmware Workstation Pro

操作系统：Ubuntu 16.04 LTS

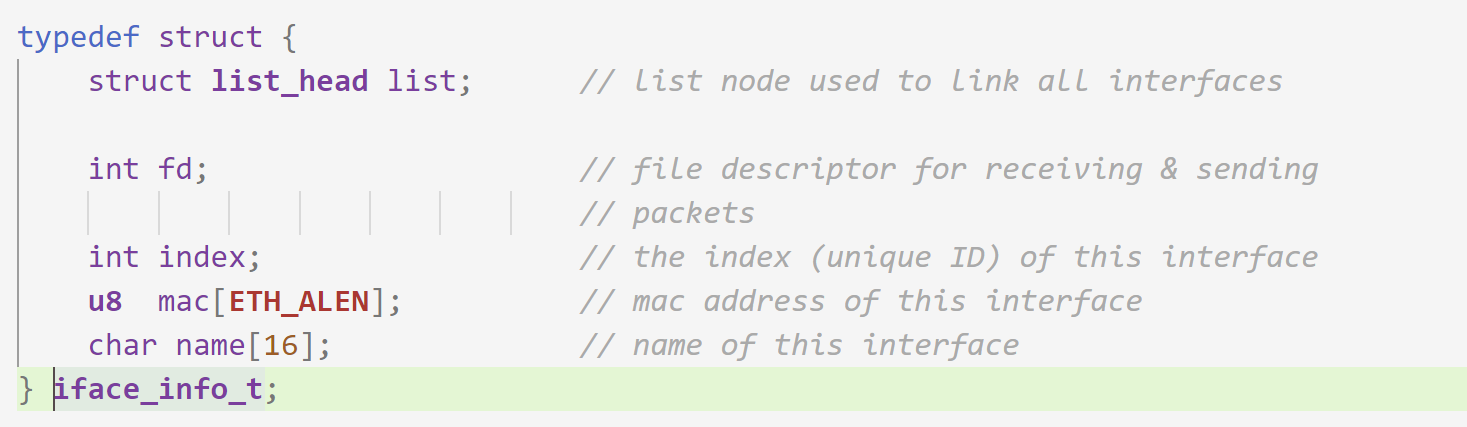
gcc：5.4.0

python：2.7.12

Mininet：2.3.0

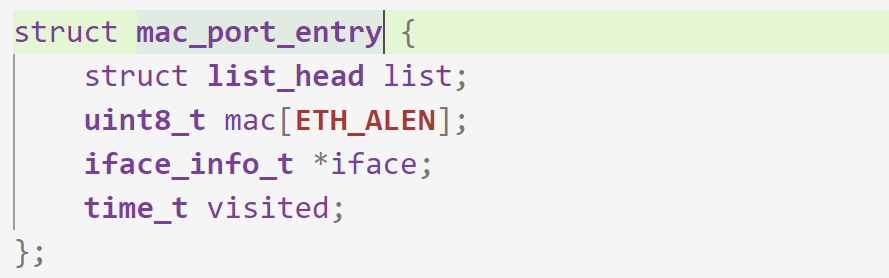
Wireshark：2.6.10

1. **实验内容**
2. 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）。
3. 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。
4. **实验流程**
5. 首先分析交换机中所包含的数据结构：
6. ：端口信息结构体：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。

1. ：mac地址与端口映射表：mac\_port\_entry



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。

1. ：mac地址与hash映射表：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。

1. 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。



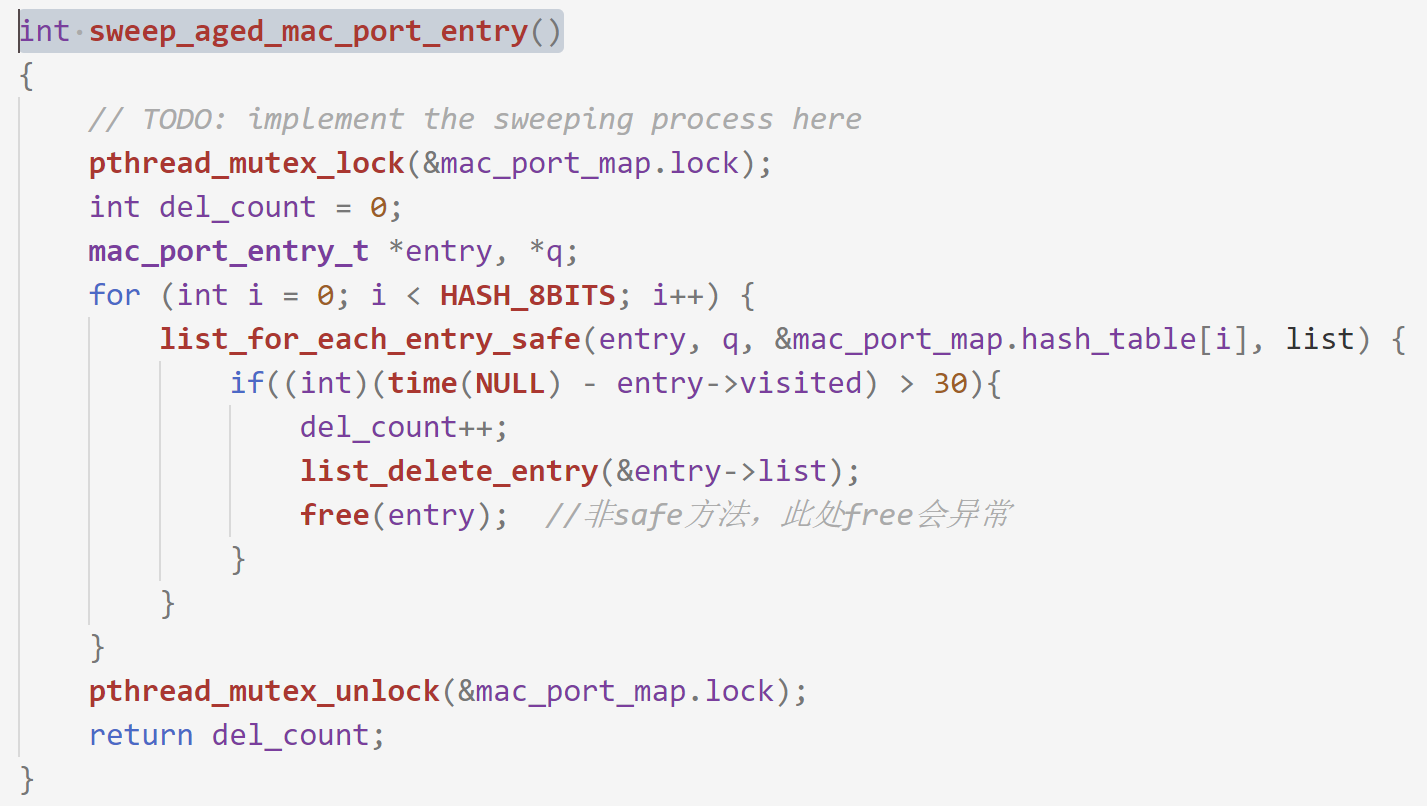
我的实现过程是首先根据mac地址计算对应的hash值，然后通过list\_for\_each\_entry宏函数对对应hash值下的链表进行遍历，遍历时加互斥锁，退出时解锁，通过findFlag作为找到对应mac地址的标志。若找到，则在退出时更新老化时间visit，并返回转发端口；若未找到，则在退出时返回null。

②. 接下来实现void insert\_mac\_port(u8 mac[ETH\_ALEN], iface\_info\_t \*iface)函数。



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。



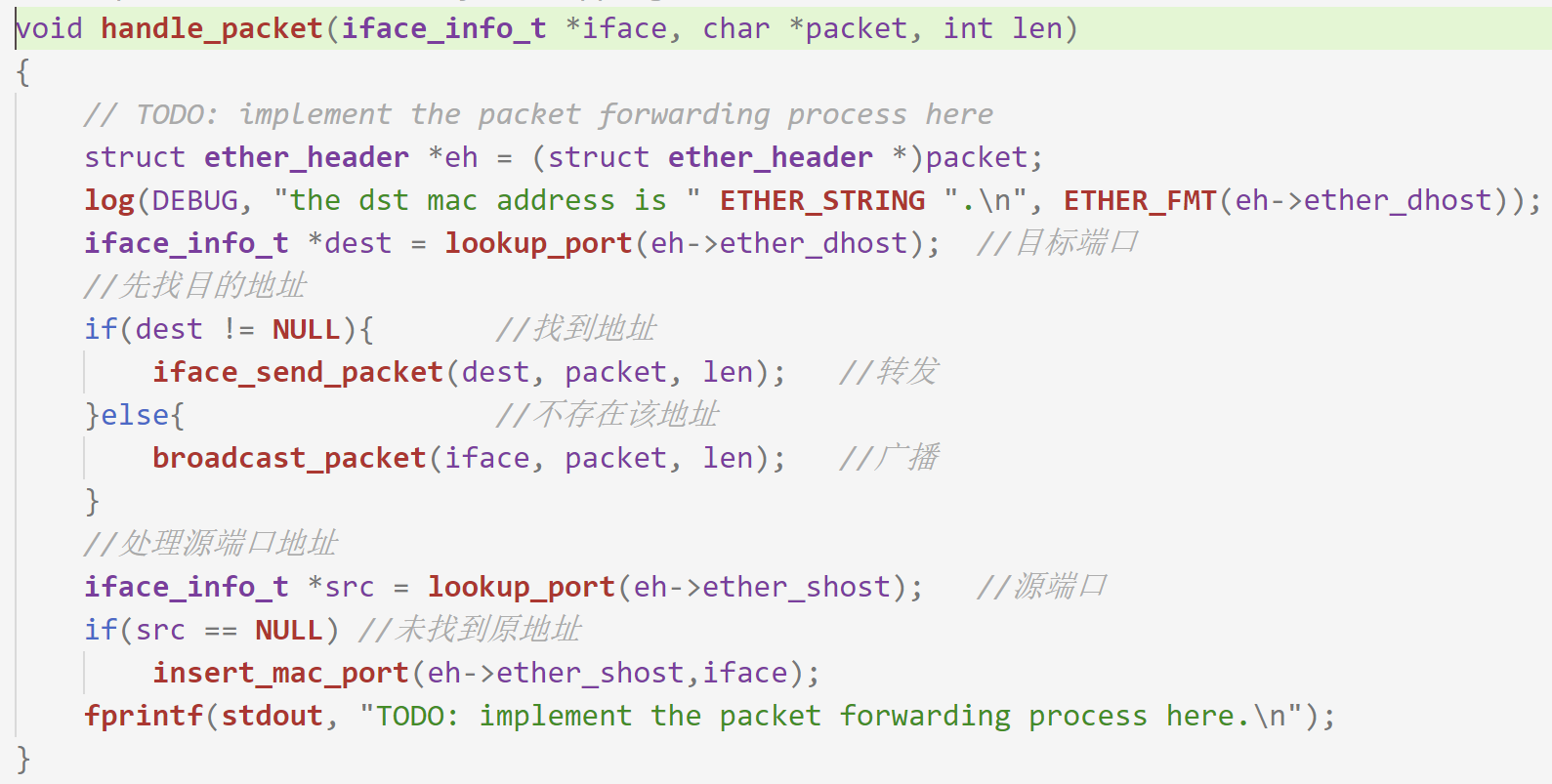
该函数的难点在于遍历并删除超过30秒未访问的转发条目，通过内核链表提供的list\_for\_each\_entry\_safe宏函数对链表进行遍历，找到与当前时间差超过30的条目并执行删除操作，删除操作使用的也是系统提供的宏函数。在调试过程中发现，执行free操作必须要使用safe的遍历方法，否则会产生指针错误。

④. 接下来是void broadcast\_packet(iface\_info\_t \*iface, char \*packet, int len)函数的实现。



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。



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。

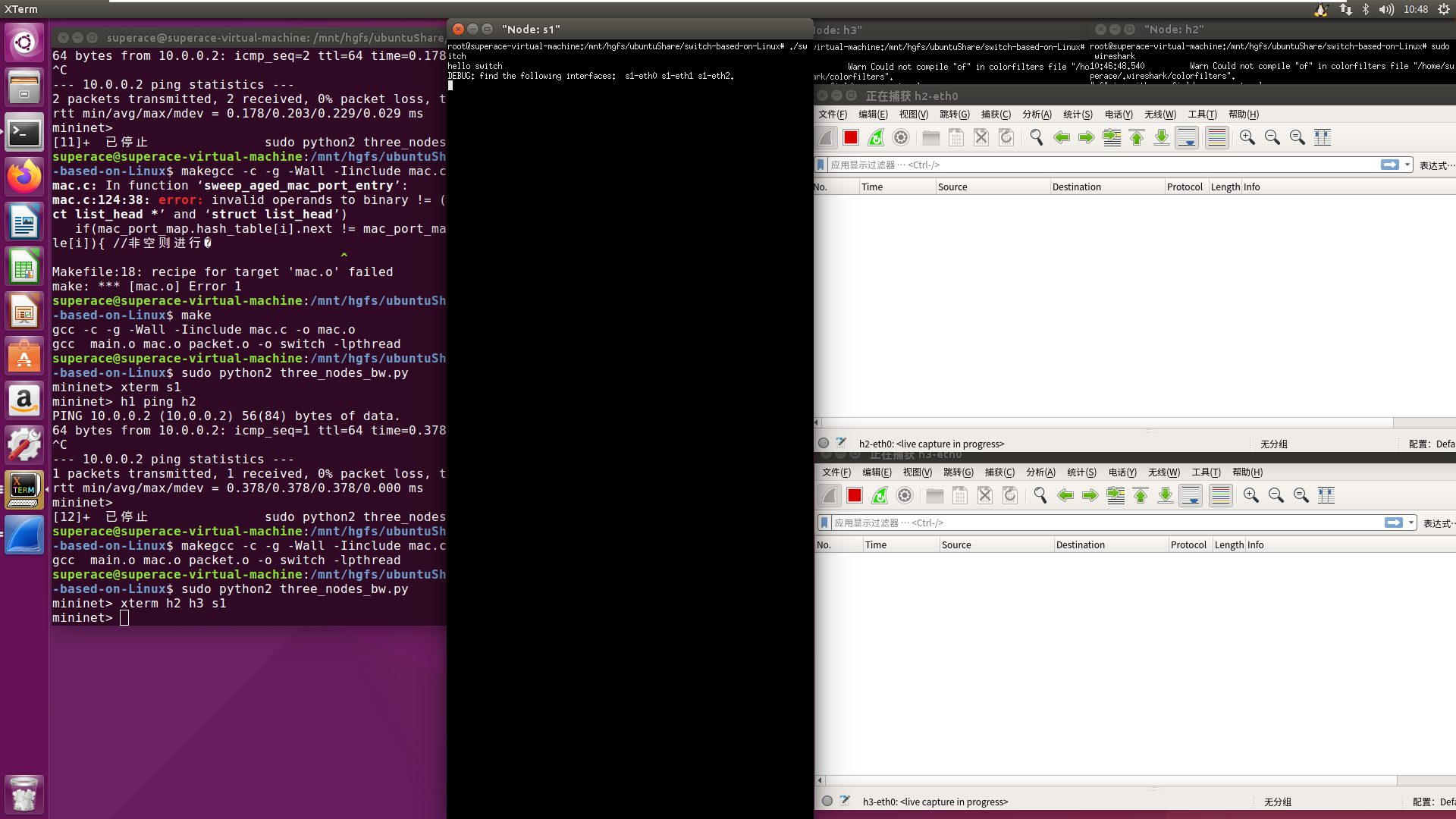


图1 实验初始化环境展示

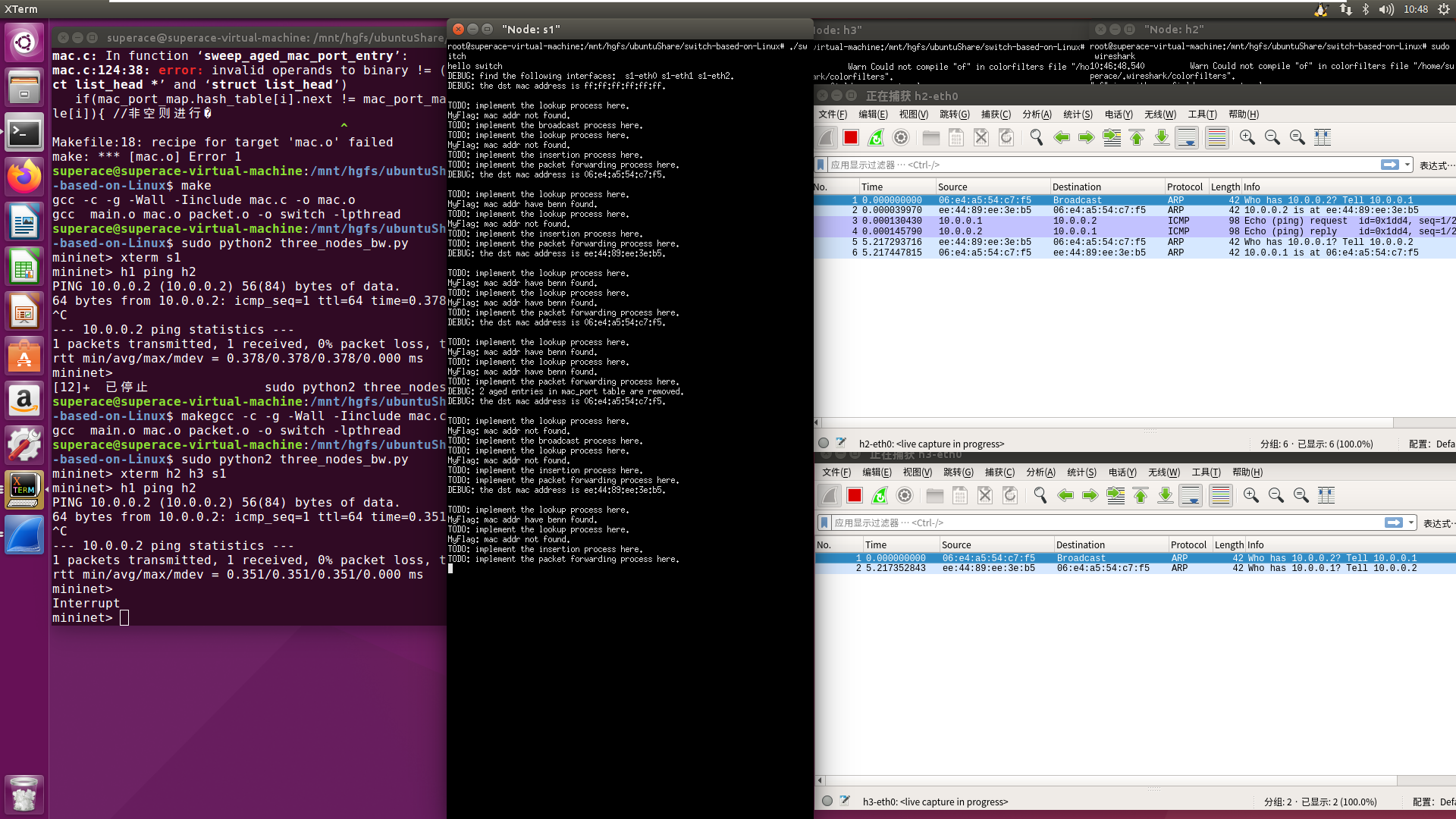


图2 h1 ping h2 实验结果

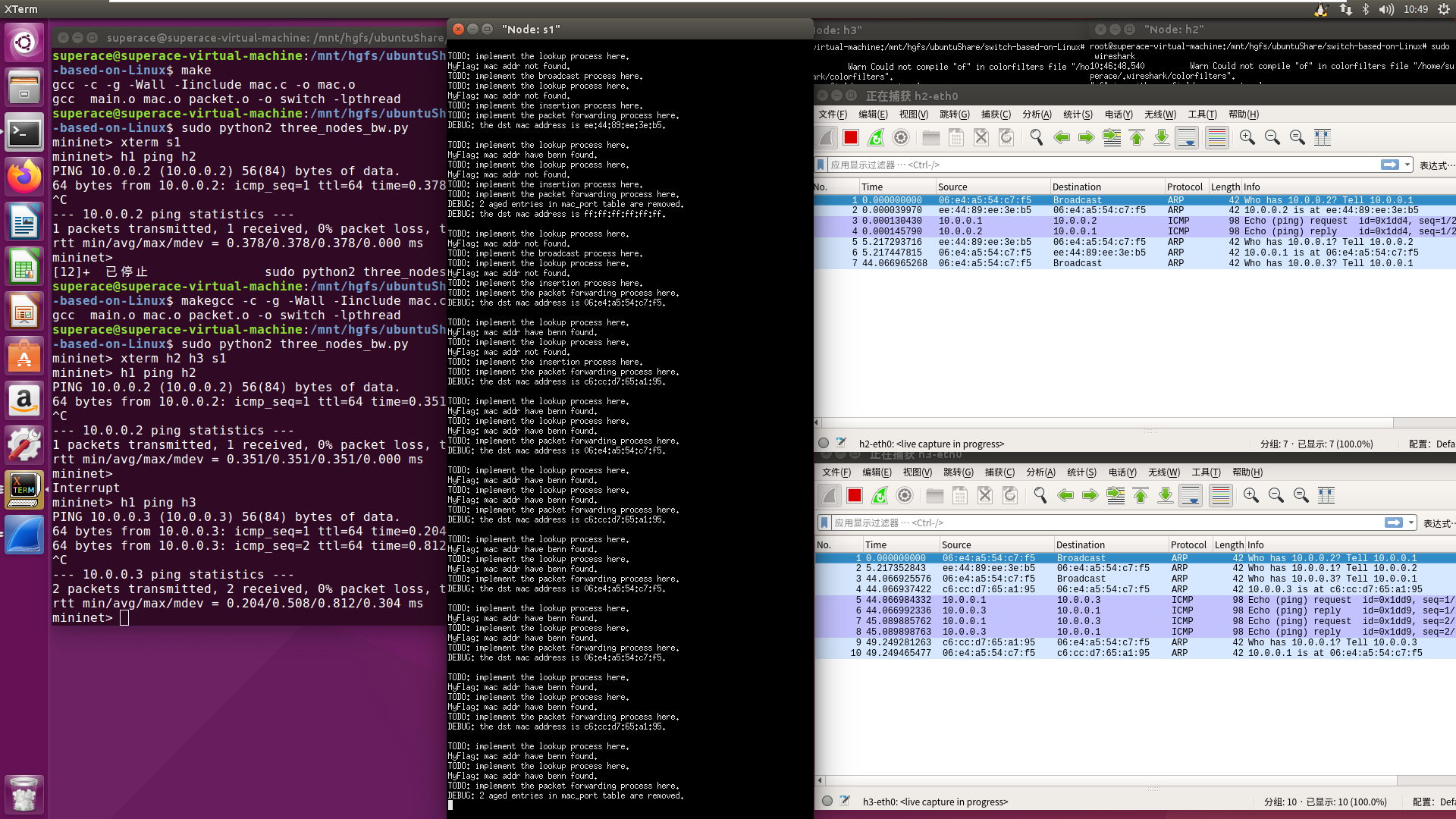


图3 h1 ping h3 实验结果

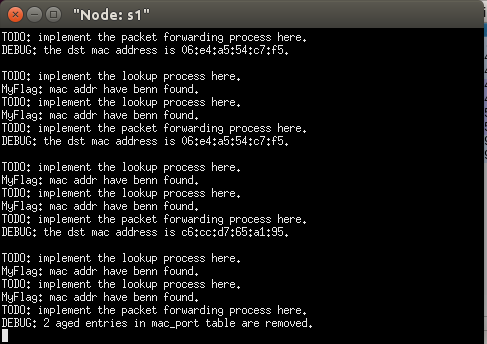


图4 s1交换机删除表项信息

1. **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。

1. **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。