

# 交换机转发实验报告

张翔雨 2018K8009929035

## 一、实验题目：交换机转发实验

## 二、实验内容

- 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

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN]);  
void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface);  
int sweep_aged_mac_port_entry();  
void broadcast_packet(iface_info_t *iface, const char *packet, int len);  
void handle_packet(iface_info_t *iface, char *packet, int len);
```

- 使用iperf和给定的拓扑进行实验，对比交换机转发与集线器广播的性能

## 三、实验过程

### 1.实现对数据结构mac\_port\_map的所有操作

#### (1) 实现lookup\_port函数

函数功能为查找mac地址对应的端口号。

函数的实现逻辑为：首先根据待查询的Mac地址查到Hash值对应的链表，遍历对应链表中是否保存有对应的Mac地址和端口号，若检查到则返回对应端口，若检索不到则返回NULL。为保证与老化线程互斥，整个查询的操作需要加锁。实现如下：

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN])  
{  
    pthread_mutex_lock(&mac_port_map.lock);  
    mac_port_entry_t *entry, *q;  
    list_for_each_entry_safe(entry, q, &mac_port_map.hash_table[hash8((char  
*)mac, sizeof(u8)*ETH_ALEN)], list)  
    {  
        if (!memcmp(entry->mac, mac, sizeof(u8)*ETH_ALEN))  
        {
```

```

        pthread_mutex_unlock(&mac_port_map.lock);
        return entry->iface;
    }
}
pthread_mutex_unlock(&mac_port_map.lock);
return NULL;
}

```

## (2) 实现insert\_mac\_port函数

函数功能为插入mac地址和port对应关系。

函数实现逻辑为：遍历整个哈希表，若找到地址，则更新该链表项的访问时间和对应端口，若未找到，则插入对应关系到对应位置。因为该操作也需要与删除老化表项的操作互斥，因此也需要加锁，代码如下：

```

void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)
{
    pthread_mutex_lock(&mac_port_map.lock);
    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(!memcmp(entry->mac, mac, sizeof(u8)*ETH_ALEN))
            {
                entry->iface = iface;
                entry->visited = time(NULL);
                pthread_mutex_unlock(&mac_port_map.lock);
                return ;
            }
        }
    }
    entry = malloc(sizeof(mac_port_entry_t));
    entry->iface = iface;
    memcpy(entry->mac, mac, sizeof(u8)*ETH_ALEN);
    entry->visited = time(NULL);
    list_add_tail(&entry->list, &mac_port_map.hash_table[hash8((char
*)mac, sizeof(u8)*ETH_ALEN)]);
    pthread_mutex_unlock(&mac_port_map.lock);
}

```

### (3) 实现sweep\_aged\_mac\_port\_entry函数

函数的功能为当转发表中有表项超过30s没有被访问过的时候，删除该表项。

函数实现逻辑较为简单，每隔1s调用该函数，检查各链表项与当前时间的差值是否超过30s，如果超过30s则删除。此操作需要与其他操作互斥，所以加锁。代码如下：

```
int sweep_aged_mac_port_entry()
{
    pthread_mutex_lock(&mac_port_map.lock);
    int n = 0;
    mac_port_entry_t *entry, *q;
    time_t now = time(NULL);
    for (int i = 0; i < HASH_8BITS; i++) {
        list_for_each_entry_safe(entry, q, &mac_port_map.hash_table[i], list) {
            if(now - entry->visited == MAC_PORT_TIMEOUT) {
                list_delete_entry(&entry->list);
                free(entry);
                n++;
            }
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);
    return n;
}
```

### (4) 实现handle\_packet函数

函数主要功能是实现收到包之后的转发工作。

函数实现逻辑是收到包之后先判断要发送的Mac地址是否在转发表中，若在，则向对应端口发出，若不在，则广播该包，最后执行插入函数，判断是否需要将源Mac地址更新到转发表中

```
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    struct ether_header *eh = (struct ether_header *)packet;
    iface_info_t *dst_iface = lookup_port(eh->ether_dhost);
    if(dst_iface)
        iface_send_packet(dst_iface, packet, len);
    else
        broadcast_packet(iface, packet, len);
    insert_mac_port(eh->ether_shost, iface);
    free(packet);
}
```

## (5) 实现broadcast\_packet函数

与上一个实验相同，复用即可

## 2.测量网络传输效率

在可执行文件所在目录下执行 `three_nodes_bw.py` 脚本启动mininet，在mininet中启动h1,h2,h3,s1四个终端，在s1中执行 `./switch` 启动hub，分别将h1作为服务器和客户端运行iperf测量网络实际带宽。

## 四、实验结果

### 1.交换机网络连通性验证

分别在h1，h2，h3三个节点中ping其他两个节点，结果如下，包括了交换机节点的debug信息以证明函数正常运行，证明节点广播网络连通性正常。

```
"Node: h2"
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# ping 10.0.0.3 -c 4
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.113 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.188 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.241 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.164 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3080ms
rtt min/avg/max/mdev = 0.113/0.176/0.241/0.046 ms
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# ping 10.0.0.1 -c 4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.154 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.164 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.209 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.251 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.154/0.194/0.251/0.038 ms
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching#

"Node: h3"
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# ping 10.0.0.1 -c 4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.141 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.417 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.283 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.169 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.141/0.252/0.417/0.108 ms
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# ping 10.0.0.2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.170 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.228 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.409 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.584 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3080ms
rtt min/avg/max/mdev = 0.170/0.347/0.584/0.162 ms
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching#

"Node: s1"
iface-port found.
DEBUG: the dst mac address is a6:86:4b:56:c0:b2.
DEBUG: the src mac address is a2:76:69:cc:10:64.
iface-port found.
DEBUG: the dst mac address is a2:76:69:cc:10:64.
DEBUG: the src mac address is a6:86:4b:56:c0:b2.
iface-port found.
DEBUG: the dst mac address is a6:86:4b:56:c0:b2.
DEBUG: the src mac address is a2:76:69:cc:10:64.
iface-port found.
DEBUG: the dst mac address is a2:76:69:cc:10:64.
DEBUG: the src mac address is a6:86:4b:56:c0:b2.
DEBUG: 1 aged entries in mac_port table are removed.

"Node: h1"
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# ping 10.0.0.3 -c 4
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.219 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.180 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.435 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.369 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3057ms
rtt min/avg/max/mdev = 0.180/0.300/0.435/0.104 ms
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# ping 10.0.0.2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.154 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.167 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.397 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.339 ms

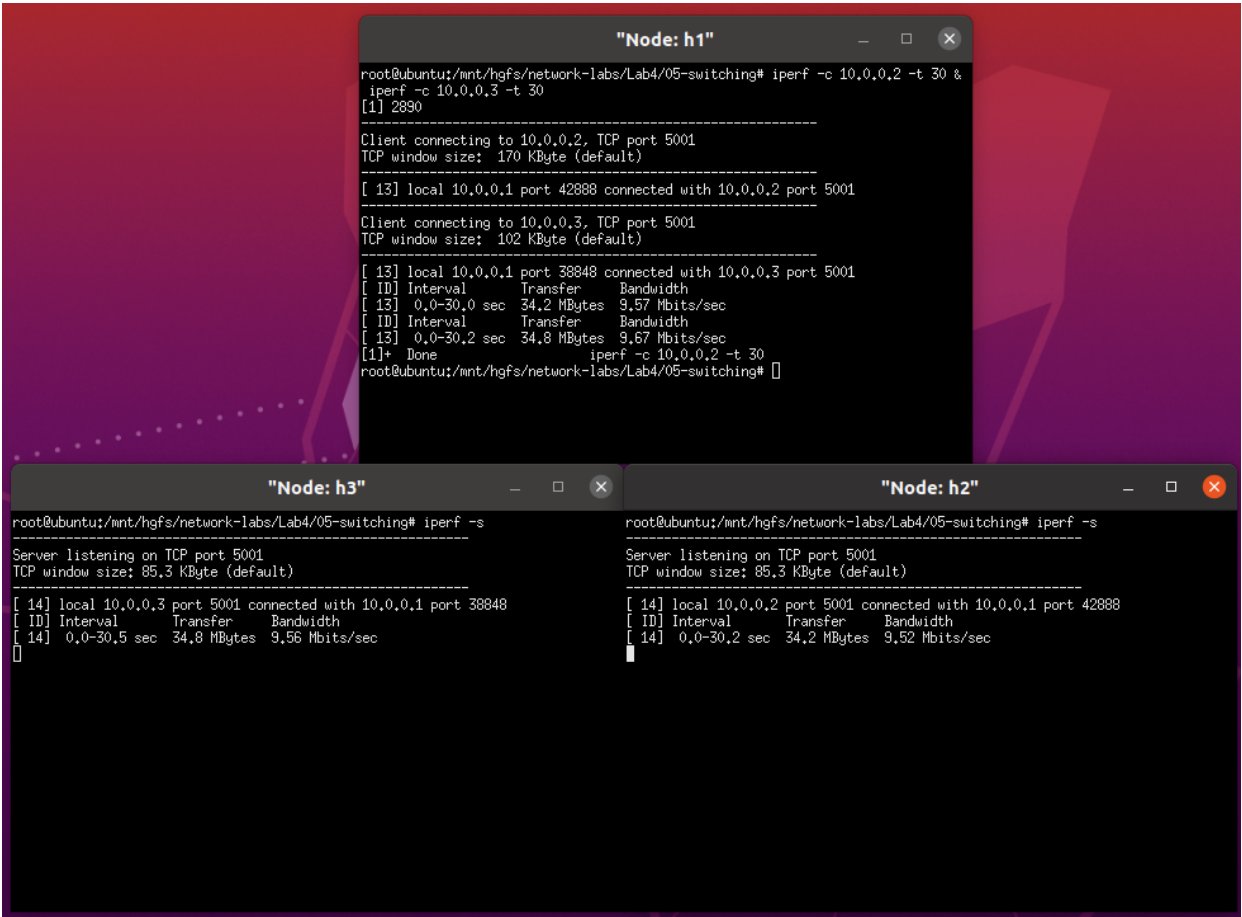
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3034ms
rtt min/avg/max/mdev = 0.154/0.264/0.397/0.105 ms
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching#
```

2.广播网络传输效率测量

(1)h1为客户端，h2，h3为服务器

|               | h1→h2 h1→h3 |      |               | h2   | h3   |
|---------------|-------------|------|---------------|------|------|
| 发送带宽(Mbits/s) | 9.57        | 9.67 | 接受带宽(Mbits/s) | 9.52 | 9.56 |
| 实际带宽(Mbits/s) | 20          | 20   | 实际带宽(Mbits/s) | 10   | 10   |
| 利用率           | 95.8%       |      |               |      |      |

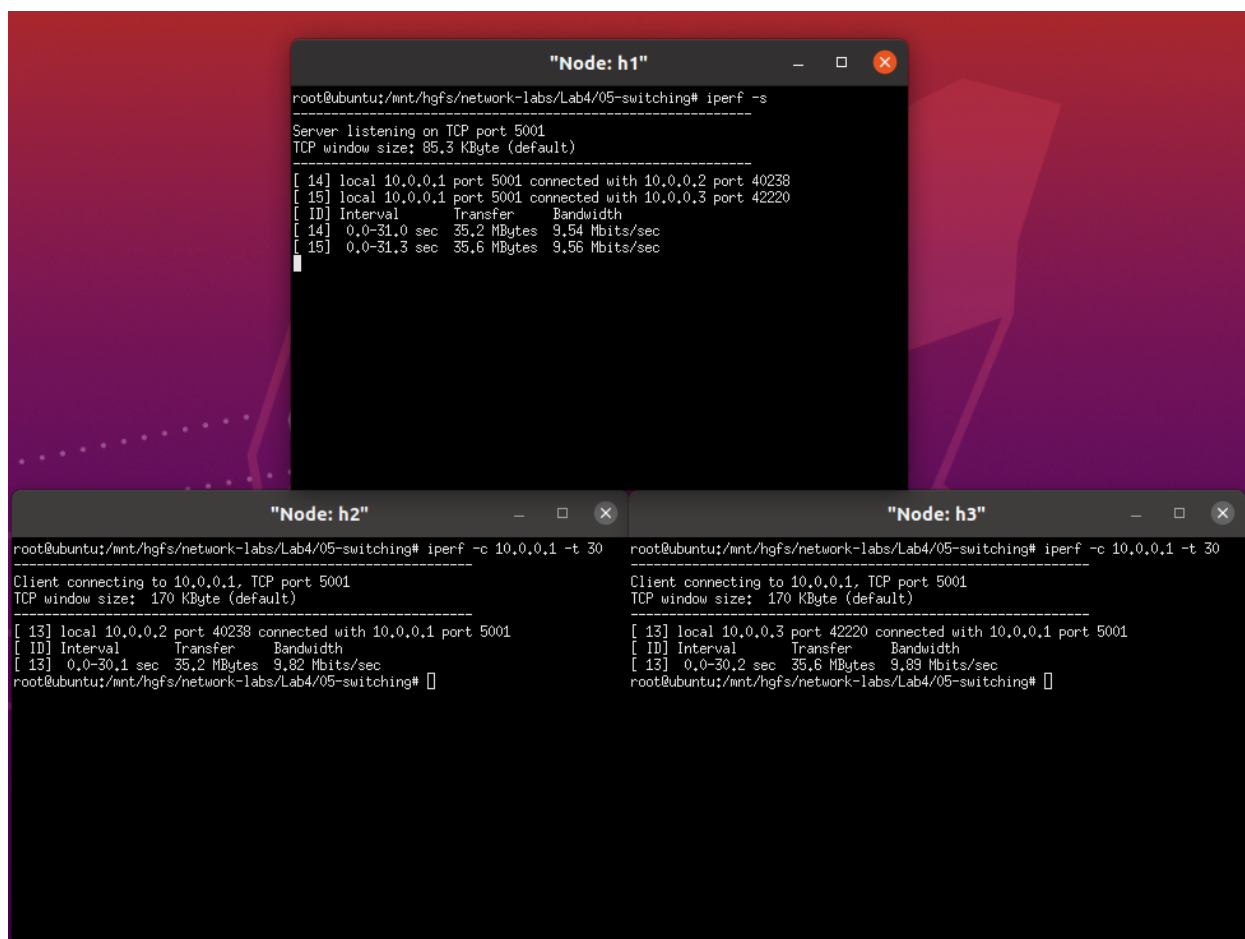
测试截图



(2)h2，h3为客户端，h1为服务器

|               | h2→h1   | h3→h1 |               | h2   | h3   |
|---------------|---------|-------|---------------|------|------|
| 接受带宽(Mbits/s) | 9.54    | 9.56  | 发送带宽(Mbits/s) | 9.82 | 9.89 |
| 实际带宽(Mbits/s) | 20      | 20    | 实际带宽(Mbits/s) | 10   | 10   |
| 利用率           | 97.025% |       |               |      |      |

测试截图



```
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 14] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 40238
[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.3 port 42220
[ ID] Interval      Transfer      Bandwidth
[ 14] 0.0-31.0 sec  35.2 MBytes  9.54 Mbits/sec
[ 15] 0.0-31.3 sec  35.6 MBytes  9.56 Mbits/sec

root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 170 KByte (default)
-----
[ 13] local 10.0.0.2 port 40238 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.1 sec  35.2 MBytes  9.82 Mbits/sec
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching#

root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 170 KByte (default)
-----
[ 13] local 10.0.0.3 port 42220 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.2 sec  35.6 MBytes  9.89 Mbits/sec
root@ubuntu:/mnt/hgfs/network-labs/Lab4/05-switching#
```

## 五、结果分析

### 网络传输效率结果对比

前一次实验h1做客户端和服务器的结果带宽利用率分别为

| 利用率      | 广播网络    | 交换机网络   |
|----------|---------|---------|
| h1→h2,h3 | 47.925% | 95.8%   |
| h2,h3→h1 | 91.425% | 97.025% |

可以看到交换机网络在两种情况下相较于广播网络利用率都有了提升，其中h1作为客户端时这种提升尤为明显。

h1 做客户端时，同时向h2,h3发包，h2和h3通过交换机将数据包发送给h1 对应的端口，而在广播网络下,则会将来自h2,h3 的数据广播，这样h2发的包会广播到h3，降低有效带宽，h3同理，因此交换机明显提升了这种情况下的利用率。

## 六、思考题

1.交换机在转发数据包时有两个查表操作：根据源MAC地址、根据目的MAC地址，为什么在查询源MAC地址时更新老化时间，而查询目的MAC地址时不更新呢？

首先明确老化时间存在的目的，因为交换机学习建立转发表的过程是将明确的Mac地址与端口号的对应关系填入表中，当源地址发来包时，交换机可以明确这个端口可以到达源地址。因此在查询源地址时可以更新老化时间。

首先查询到目的Mac地址时不能保证当前端口依然能送达，因此此时不需要更新，如果此时更新了老化时间且一段时间内该地址没有再发包到交换机，同时此主机切换了在交换机上的对应端口，则交换机会一直将发往该地址的包发向旧的错误的端口，而正确操作应该是老化后只能广播以找到新的正确的端口。

2.网络中存在广播包，即发往网内所有主机的数据包，其目的MAC地址设置为全0xFF，例如ARP请求数据包。这种广播包对交换机转发表逻辑有什么影响？

目的MAC地址设置为全0xFF的包即为广播包，交换机收到这种包的时候一定会广播，同时记录下源地址和端口的对应关系。通常这种广播包被用于主机A想找到主机B但不知道目标地址时使用，当B接受到广播包时会发送回应包，在这个过程中，交换机的转发表建立了A-B之间的转发表，下次两台主机的通信就可以通过建立的转发表进行通信了。

3.理论上，足够多个交换机可以连接起全世界所有的终端。请问，使用这种方式连接亿万台主机是否技术可行？并说明理由。

不可行。

- 首先安全性就难以保证，这种网络连接起来是一个巨大的局域网，会很容易受到网络黑客的攻击，数据安全收到威胁。
- 其次世界规模的交换机网络，会产生巨大的转发表资源消耗，如果转发表较小，其中的表项会不断地老化或被替换掉，效率极低。同时这种一维的交换网络拓扑结构也会使得许多基于网络拓扑的算法由于规模过大难以实现。

## 七、实验总结

本次实验有一点代码量，需要整体思考整个交换机工作机制，不再像前面实验只需实现一个函数。许多在实验中没有考虑到或弄明白的问题通过思考题的导向也明白了很多。