

# 交换机转发实验

学号：2016K8009908007

姓名：薛峰

## 一、实验内容

### 1、实现交换机学习算法

①查询操作：每收到一个数据包，根据目的 MAC 地址查询相应转发条目：如果查询到对应条目，则根据相应转发端口转发数据包，并更新访问时间；否则，广播该数据包；

②插入操作：每收到一个数据包，如果其源 MAC 地址在转发表中，更新访问时间；否则，将该地址与入端口的映射关系写入转发表；

③老化操作：每秒钟运行一次老化操作，删除超过 30 秒未访问的转发条目。

### 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

```
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);  
  
svoid handle_packet(iface_info_t *iface, char *packet, int len);
```

### 3、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

## 二、实验流程

### 1、iface\_info\_t \*lookup\_port(u8 mac[ETH\_ALEN])函数

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN])  
{  
    // TODO: implement the lookup process here  
    // fprintf(stdout, "TODO: implement the lookup process here.\n");  
  
    pthread_mutex_lock(&mac_port_map.lock);  
  
    mac_port_entry_t *entry = NULL;  
  
    list_for_each_entry(entry, &mac_port_map.hash_table[hash8((char *)mac, ETH_ALEN)], list) {  
        if( memcmp(entry->mac, mac, ETH_ALEN*sizeof(u8)) == 0 ) {  
            entry -> visited = time(NULL);  
  
            // Don't Forget to release the lock !!!  
            pthread_mutex_unlock(&mac_port_map.lock);  
            return entry->iface;  
        }  
    }  
  
    pthread_mutex_unlock(&mac_port_map.lock);  
  
    return NULL;  
}
```

首先根据 mac 地址找到对应的 hash\_table，然后便利该 hash\_table，如果找到与该 mac 对应的项，则返回该项的 iface，并更新访问时间。应注意的是在返回前应该释放 lock。

## 2、void insert\_mac\_port(u8 mac[ETH\_ALEN], iface\_info\_t \*iface)函数

```
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");

    pthread_mutex_lock(&mac_port_map.lock);

    mac_port_entry_t *entry = malloc(sizeof(mac_port_entry_t));
    memcpy(entry->mac, mac, ETH_ALEN*sizeof(u8));
    entry->iface = iface;
    entry->visited = time(NULL);

    list_add_tail(&entry->list, &mac_port_map.hash_table[hash8((char *)mac, ETH_ALEN)]);

    pthread_mutex_unlock(&mac_port_map.lock);
}
```

将该 mac 和 iface 插入到 hash\_table 中。

## 3、int sweep\_aged\_mac\_port\_entry()函数

```
int sweep_aged_mac_port_entry()
{
    // TODO: implement the sweeping process here
    //fprintf(stdout, "TODO: implement the sweeping process here.\n");

    pthread_mutex_lock(&mac_port_map.lock);

    mac_port_entry_t *entry, *q;

    int i = 0;
    int count = 0;

    for (i = 0; i < HASH_8BITS; i++) {
        list_for_each_entry_safe(entry, q, &mac_port_map.hash_table[i], list) {
            if( (time(NULL) - entry->visited) >= MAC_PORT_TIMEOUT) {
                list_delete_entry(&entry->list);
                free(entry);
                count ++;
            }
        }
    }

    pthread_mutex_unlock(&mac_port_map.lock);

    return count;
}
```

便利所有 hash\_table 中的所有表项，若其访问时间为 30s 以前，则删除该表项。

## 4、void broadcast\_packet(iface\_info\_t \*iface, const char \*packet, int len)函数

```
void broadcast_packet(iface_info_t *iface, char *packet, int len)
{
    // TODO: implement the broadcast process here
    //fprintf(stdout, "TODO: implement the broadcast process here.\n");

    iface_info_t *entry = NULL;
    list_for_each_entry(entry, &instance->iface_list, list) {
        if (entry->index != iface->index)
            iface_send_packet(entry, packet, len);
    }
}
```

## 5、void handle\_packet(iface\_info\_t \*iface, char \*packet, int len)函数

```
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    struct ether_header *eh = (struct ether_header *)packet;
    //log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_dhost));
    //log(DEBUG, "the src mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_shost));

    // TODO: implement the packet forwarding process here
    //fprintf(stdout, "TODO: implement the packet forwarding process here.\n");

    iface_info_t* d_iface = lookup_port(eh->ether_dhost);
    iface_info_t* s_iface = lookup_port(eh->ether_shost);

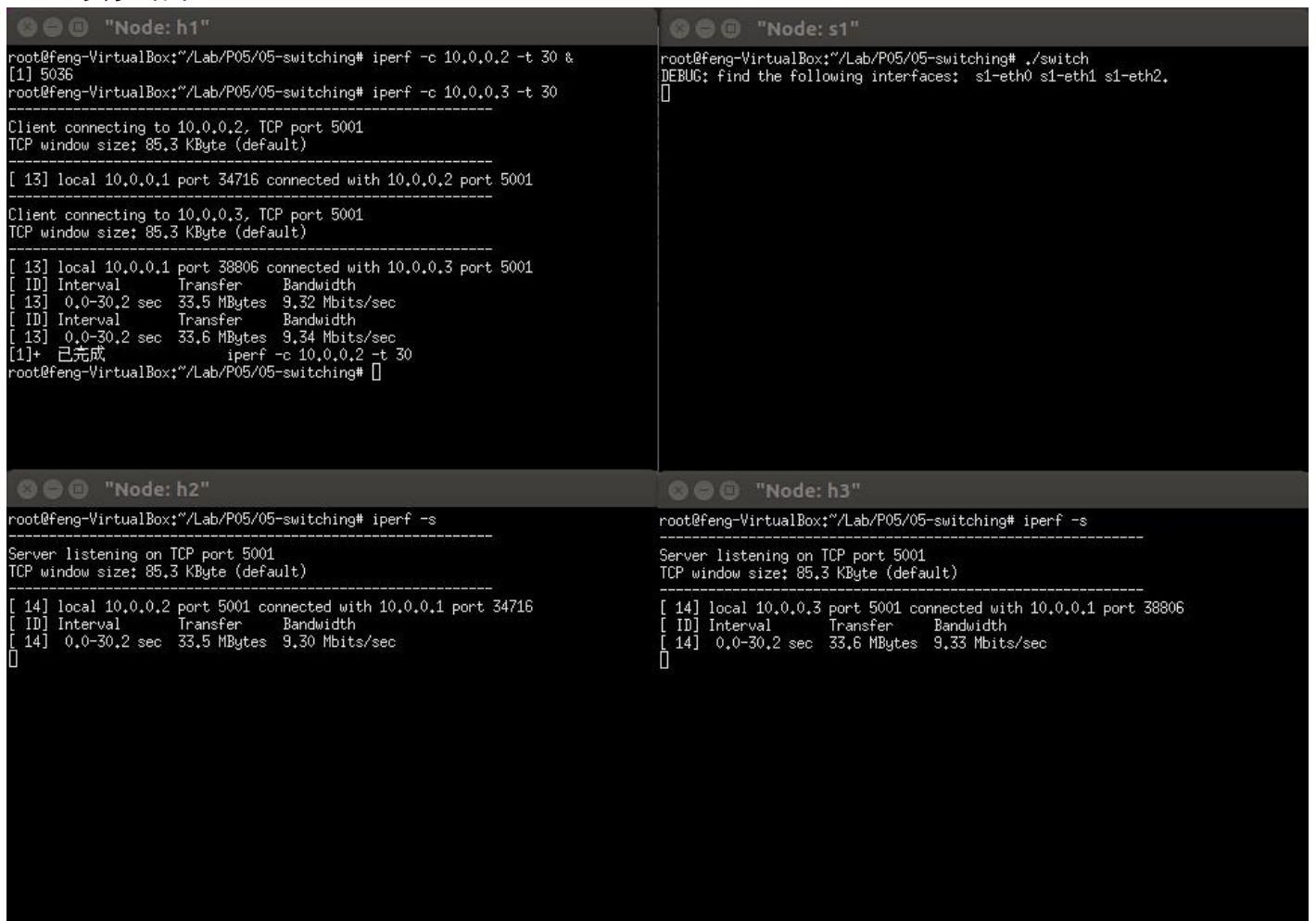
    if(d_iface != NULL)
        iface_send_packet(d_iface, packet, len);
    else
        broadcast_packet(iface, packet, len);

    if(s_iface == NULL)
        insert_mac_port(eh->ether_shost, iface);
}
```

首先查询目的 mac 地址是否在转发表中，如果存在则更新，则发送到返回的 iface，如果不存在则广播该数据包。同时查询源 mac 地址是否在转发表中，如果在则更新（这一步在 lookup\_port 函数中实现）访问时间，若不存在则添加该映射到转发表中。

## 三、实验结果及分析

### 1、实验结果



The image displays four terminal windows from a VirtualBox environment, showing the execution of iperf tests on a network topology with four nodes: h1, s1, h2, and h3.

- Node: h1**: Shows two client connections. The first connects to 10.0.0.2 port 5001, and the second connects to 10.0.0.3 port 5001. Both show a bandwidth of approximately 9.32-9.34 Mbits/sec over a 30-second interval.
- Node: s1**: Shows the output of the ./switch command, listing interfaces s1-eth0, s1-eth1, and s1-eth2.
- Node: h2**: Shows a server listening on TCP port 5001. It receives a connection from 10.0.0.2 port 5001, showing a bandwidth of 9.30 Mbits/sec.
- Node: h3**: Shows a server listening on TCP port 5001. It receives a connection from 10.0.0.3 port 5001, showing a bandwidth of 9.33 Mbits/sec.

H1: iperf client; H2, H3: iperf servers

```
"Node: h1"
root@feng-VirtualBox:~/Lab/P05/05-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 10.0.0.3 port 42924
[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 41628
[ 10] Interval      Transfer      Bandwidth
[ 14] 0.0-30.6 sec  33.8 MBytes  9.26 Mbits/sec
[ 15] 0.0-30.7 sec  34.1 MBytes  9.32 Mbits/sec
[ 16]

"Node: s1"
root@feng-VirtualBox:~/Lab/P05/05-switching# ./switch
DEBUG: find the following interfaces: s1-eth0 s1-eth1 s1-eth2.
[]

"Node: h2"
root@feng-VirtualBox:~/Lab/P05/05-switching# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.2 port 41628 connected with 10.0.0.1 port 5001
[ 10] Interval      Transfer      Bandwidth
[ 13] 0.0-30.2 sec  34.1 MBytes  9.48 Mbits/sec
root@feng-VirtualBox:~/Lab/P05/05-switching# █

"Node: h3"
root@feng-VirtualBox:~/Lab/P05/05-switching# iperf -c 10.0.0.1 -t 30
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.3 port 42924 connected with 10.0.0.1 port 5001
[ 10] Interval      Transfer      Bandwidth
[ 13] 0.0-30.1 sec  33.8 MBytes  9.41 Mbits/sec
root@feng-VirtualBox:~/Lab/P05/05-switching# █
```

H1: iperf server; H2, H3: iperf clients

## 2、结果分析

### ■ h1: iperf client; h2, h3: iperf servers

**结果：**对于交换机学习算法，当 h1 同时向 h2 和 h3 发送数据时，带宽都是接近 10Mb/s；然而对于广播算法，当 h1 同时向 h2 和 h3 发送数据是，带宽减半。

**分析：**这是由于对于交换机学习算法而言，当 h1 发送一个包到交换机时，该包只会发送给目的 mac 地址；而对于广播算法，当 h1 发送一个包到 hub 后，hub 会分别向 h2 和 h3 发送数据。因此，对于交换机算法，switch 和 h2 的链路上只有 h1 发给 h2 的数据，多以其带宽能接近于 10Mb/s；然而 switch 和 h2 的链路上不仅有 h1 发给 h2 的数据也有 h1 发给 h3 的数据，因此其带宽减半。

### ■ h1: iperf server; h2 h3: iperf clients

**结果：**交换机学习算法和广播算法结果相同，带宽均接近于 10Mb/s。

**分析：**对于交换机学习算法，当 h2 或 h3 发送数据到 switch 时，switch 只会向 h1 发送数据，因此每条链路上的带宽都能接近最大。对于广播算法，当 h2 发送数据到 switch 时，switch 会同时向 h1 和 h3 发送数据，但是 switch 向 h3 发送数据不影响 h3 向 switch 发送数据，因此每条链路的带宽也能达到最大。